University of Alberta

High-Performance and Compact Gaussian Noise Generators, Fading Channel Simulators, and Layered Space-Time Decoders

by

Amirhossein Alimohammad

©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Department of Electrical and Computer Engineering

Edmonton, Alberta
Spring 2007

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-29645-5
Our file Notre référence
ISBN: 978-0-494-29645-5

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

To my mother, Homa Sattari,

and

the memory of my father, Abolghasem Alimohammad

for all their love and support

Abstract

The quest for higher data rates has led to significantly more complex physical layer signal processing algorithms. This complexity impacts the design and implementation of wireless communication systems in two different ways. First, the great computational demands can exceed the performance available from even high-end processors. Hence, it is important to investigate architectural techniques that facilitate the implementation of the necessary real-time signal processing instead of using a faster, but increasingly expensive and more power-hungry, conventional processors. Second, significant design effort must be dedicated to simulating and verifying alternative signal processing algorithms. For example, the Monte Carlo (MC) simulation technique is commonly used to evaluate the bit error-rate (BER) performance of these systems. While software simulations are widely used in the design and verification of communication systems, the required MC simulation times are now becoming prohibitively long. Fortunately, hardware-based techniques can speed up simulation by several orders of magnitude. Hardware-based simulation can also help to identify implementation bottlenecks and calculate design metrics for different candidate algorithms at early stages in the design.

Since the bit error rate of wireless communication systems strongly depends on radio channel characteristics, it is important that the chosen channel model reproduce the statistical properties of the real world channel as faithfully as possible. Moreover, since attenuation at the receiver is commonly modeled as a sequence of variates with a Gaussian probability distribution function (PDF), random variates near the center of the distribution do not contribute significantly to the probability of error in low bit error rate systems. Hence, to obtain accurate BER results in simulation, the PDF of generated noise must be especially close to the true Gaussian PDF at the tails of the PDF.

This thesis reports significant progress in several key building blocks used in the design and evaluation of wireless communication systems as follows:

- The design and implementation of the fastest and most compact disclosed hardware Gaussian variate generator with accurate statistical properties.
- An improved hardware fading channel model based on the sum-of-sinusoids (SOS) approach and its implementation.
- An accurate and compact implementation of a parameterized hardware fading channel simulator using digital filters.
- Mapping and implementation of the layered space-time decoding algorithm onto a moderately-parallel and scalable single-instruction multiple-data (SIMD) processor architecture that was developed at the University of Alberta.

Acknowledgements

The road to my Ph.D. has been a long journey spanning three continents and more years than I would like to count. I started this journey at Sharif University of Technology in Tehran, Iran. I took a detour to Germany and finally ended up in Edmonton at the University of Alberta. Now that I am at the very end of my Ph.D. and about to start new adventures, I would like to think back on the many people who helped to make the road so exciting and memorable.

Prof. Bruce F. Cockburn has been enthusiastic, supportive, insightful, and a superb mentor. I feel very lucky to have had the opportunity to work under his supervision.

Prof. Christian Schlegel has given me many opportunities to expand my horizons and helped me enter new research domains.

My committee members Prof. José Nelson Amaral and Prof. Vincent Gaudet provided valuable input on my dissertation. I really enjoyed discussing my work with Prof. Steven J. Wilton, who was an excellent external examiner.

I would like to thank many friends in the VLSI and HCDC labs for making the years so enjoyable. I have had fun doing many activities from paintball to Carcassonne to discussing the newest trends in digital design. I have spent countless days and hours in deep discussion with Saeed Fouladi Fard designing and optimizing many projects that cross between wireless communication and digital design. I am very grateful for all his help and collaboration. Tyler L. Brandon helped in ASIC design flow, Steven J. Dillen and Daniel A. Leder provided constructive discussions on DSP-RAM architecture, Leendert van den Berg edited many emails and letters, and John C. Koob showed me how fun Latex can be. I have enjoyed getting to know Sheryl, Sheehan, Behnam, Ram, Maz, and many other friends in the department.

As my specific knowledge in digital communications was growing, Terra and her family encouraged me to try new activities such as cross-country skiing, white-water canoeing, and sleeping in snow caves.

My family has always believed in me and inspired me to do my best. Elham has made the many kilometers between Edmonton and Tehran much less by emailing me every day. My mom has always been there to help and support all my endeavours. Her love has inspired and helped me to reach my goals. When I began school, my dad helped me with math and science and showed me that when I learned something, I was not at the end of my knowledge, but on the doorstep getting a small glimpse of all that I do not know. I still feel this today.

Many thanks to all the people who have helped to make the journey of my Ph.D. extraordinary.

Table of Contents

1 Motivation 1
- 1.1 Thesis Contributions . . . 3
- 1.2 Thesis Outline . . . 5

2 Introduction 7
- 2.1 Background . . . 8
  - 2.1.1 Frequency Domain Representations of Signals . . . 8
  - 2.1.2 Random Processes . . . 9
  - 2.1.3 Linear Systems Response to Random Signals . . . 15
  - 2.1.4 Baseband Processing . . . 18
  - 2.1.5 Geometric View of Signals and Transformations . . . 24
- 2.2 Noise Models at the Receiver . . . 26
- 2.3 Wireless Channel Effects . . . 30
- 2.4 High-Throughput Wireless Communication Systems . . . 34
- 2.5 Algorithm Efficiency . . . 36
- 2.6 Base-Band Signal Processing Platforms . . . 38
- 2.7 Rapid Prototyping in Digital Design . . . 42

3 Gaussian Variate Generators 46
- 3.1 Gaussian Distribution, Algorithms, and Related Work . . . 49
- 3.2 A Closer Look at PNGs . . . 53
- 3.3 Implementation of Trigonometric Functions . . . 61
- 3.4 Gaussian Variate Generator Implementation . . . 64
  - 3.4.1 Implementation of a GVG Using Iterative Algorithms . . . 65
  - 3.4.2 A GVG Using Non-Uniform Quantization and Table Lookup . . . 68
  - 3.4.3 Implementation of a GVG Using Piecewise Polynomial Curve Fitting 71
- 3.5 The GVG Statistical Tests . . . 78
- 3.6 Conclusions . . . 82

4 SOS-based Fading Channel Simulators 85
- 4.1 Parameters of Fading Channels . . . 88
- 4.2 Channel Models . . . 90
- 4.3 Stochastic Models for Fading Channels . . . 95
- 4.4 Analysis of SOS-Based Fading Channel Models . . . 98
- 4.5 Implementation of an SOS Fading Channel Simulator . . . 106
- 4.6 An Improved SOS-based Fading Channel Emulator . . . 116

4.7 Conclusions 123

5 Filter-Based Fading Channel Simulators **125**
5.1 Digital Filter Structures and Quantization Error Effects 128
5.2 Generation of Correlated Random Sequences 132
5.3 Constraints on Filter Design 134
5.4 Filter Design for Fading Channel Simulators 141
5.5 Implementation and Statistical Verification 143
5.6 A Flexible Filter Processor for Fading Channel Emulation 149
5.7 Conclusions 155

6 An Efficient Parallel Architecture for LST Decoding in MIMO Systems **156**
6.1 Mathematical Model of MIMO Systems 159
6.2 Algorithms for Systems of Linear Equations 162
6.3 MIMO Decoding Techniques 164
6.4 A Parallel Architecture for Digital Signal Processing 172
6.5 Mapping LST Decoding onto DSP-RAM 177
6.6 Implementation of a MIMO Receiver 180
6.7 Conclusions 185

7 Conclusions and Future Work **188**
7.1 Future Work 191
7.1.1 An Alternative Gaussian Variate Generator 191
7.1.2 On the Ergodicity of the SOS-based Fading Channel Models 194
7.1.3 Architecture of a Wireless Channel 195
7.1.4 Joint Channel Estimation and Symbol Decoding for LST Decoders 198

Bibliography **202**

A List of Publications Arising From Thesis **218**

List of Tables

2.1 Parameters of different Xilinx FPGAs. 43
2.2 Parameters of Altera Stratix EP1S80F1508C6 FPGA. 44

3.1 Maximum absolute value of a GV for various precisions of u_1. 53
3.2 Published FPGA Implementations of a GVG. 53
3.3 Performance of three CTGs implemented on the Xilinx Virtex-II XC2V4000-6. 60
3.4 PNGs used in published GVG designs. 60
3.5 Typical realizations of the new GVG. 77

4.1 Implementation of the fading channel simulator on different FPGAs. 111
4.2 Implementation of the fading channel simulator on different FPGAs. 116
4.3 Maximum step size δ_o. 120
4.4 Implementation of the fading channel simulator on three different FPGAs. . 122

5.1 Some commercially available fading simulators. 126
5.2 Out-of-order execution scheme for the cascade structure. 148
5.3 Out-of-order scheduling and register renaming for the cascade structure. . . 148

6.1 Comparison of PIM architectures. 172
6.2 Processor specifications. 180
6.3 DSP-RAM PE implementation characteristics. 181
6.4 Cycle counts to decode received ST signals and calculate nulling vectors of a 4×4 channel matrix. 184
6.5 Decoding throughput of the LST decoder implementations for a (4,4) MIMO system utilizing 16-QAM modulation, with the block-fading channel model assumption, and $T_b = 100$ ms. 186

List of Figures

2.1 (a) BPSK modulator. (b) QAM modulator. 22
2.2 Two different constellations. 22
2.3 (a) PSD of white noise. (b) ACF of white noise. 27
2.4 (a) PSD of bandlimited white noise. (b) ACF of bandlimited white noise. . 28
2.5 Normalized Gaussian PDF. 29
2.6 Typical simulated Rayleigh fading at the receiver. 31
2.7 Block diagram of wireless communication systems. 34
2.8 Ergodic capacities of uncorrelated multiple antenna systems for SNR=18 dB. 36

3.1 Plot of $Q(x)$. 50
3.2 The two-dimensional distribution of 10^7 PNs pairs (u_i, u_{i+1}) generated with MLCG recursion with $x_0 = 314519$, $a = 16807$, and $m = 2^{31} - 1$, when a small portion of u_i-axis is magnified. 55
3.3 Fibonacci implementation of a 4-bit LFSR. 57
3.4 Rule 90 for cellular automata. 58
3.5 The cosine approximation error: (a) Ideal cosine function; (b) Squared error for the table look-up approximation; (c) Squared error for the first order approximation. 64
3.6 52-bit LFSR design. 67
3.7 Gaussian PDF compared with PDF generated noise samples. 68
3.8 Plot of $f(u_1) = \sqrt{-2\ln(u_1)}$. 69
3.9 Non-uniform quantization of $(0, 1)$. 69
3.10 GVG datapath using non-uniform quantization and table lookup. 70
3.11 Ideal Gaussian and generated PDFs plotted on a logarithmic scale. 71
3.12 Segmentation of $u_1 \in (0, 1)$. 73
3.13 The datapath for calculating $f(\cdot)$ function. 75
3.14 The tree structure of leading zeros counting. 76
3.15 Logic diagram for leading one detector. 77
3.16 The 0.126 mm^2 GVG chip layout in 90-nm CMOS technology. 78
3.17 Gaussian PDF compared with the PDF 10^{11} generated GVs. 79
3.18 Inverse CDF of the generated GVs. 80
3.19 Plot of n_1 versus n_2. 80
3.20 Statistical dependency of n_i and n_{i+1} for 10^7 generated GVs. 83
3.21 Autocorrelation among n_i and $n_{i\pm 2048}$ for 10^7 generated GVs. 83

4.1 Architecture of a frequency-selective fading simulator. 93
4.2 The PSD for a Rayleigh fading channel with different Doppler frequencies. 98

4.3 The zero-order Bessel function of the first kind. 99
4.4 Symmetry of receiving sinusoids in Jake's design. 101
4.5 The ACF and CCF are calculated by averaging over 10 frames of 10^5 fading samples each with $f_D T_s = 0.01$. 103
4.6 The normalized LCR calculated using 10^5 fading samples with $f_D T_s = 0.01$. 104
4.7 Envelope PDF for Zheng and Xiao's SOS fading channel model. 105
4.8 Envelope CDF for Zheng and Xiao's SOS fading channel model. 105
4.9 ACF and CCF for one block containing 10^7 fading samples using the Jakes SOS model from equation (4.10), $f_D T_s = 0.02$, $N = 8$. 106
4.10 (a) Circuit for summing $N = 2$ complex oscillators. (b) Tree-structured adder for summing $N = 8$ oscillators. 108
4.11 The PDF of the simulated fading envelope and their references with normalized Doppler rate $f_D T_s = 0.01$. 109
4.12 ACF and CCF of the quadrature component. 109
4.13 The normalized LCR of the generated fading samples. 110
4.14 The CDF of the simulated fading envelope and their references. 110
4.15 Layout of the 500 MHz semicustom fading channel variate generator. . . . 111
4.16 Datapath for generating one low-frequency oscillator. 112
4.17 The ACF calculated by averaging over 10 frames of 10^5 fading samples with $f_D T_s = 0.01$. 113
4.18 The CCF calculated by averaging over 10 frames of 10^5 fading samples with $f_D T_s = 0.01$. 113
4.19 Envelope pdf of 10^7 generated fading variates with $f_D T_s = 0.01$ for three different precisions. 114
4.20 The cdf of 10^7 generated fading variates with two different precisions. . . . 114
4.21 The normalized LCR of 10^7 generated fading envelopes with two different precisions. 115
4.22 Pipelined datapath for generating one low-frequency oscillator. 116
4.23 (a) Circuit for summing two complex oscillators. (b) Tree-structured adder for summing $N = 8$ oscillators. 116
4.24 ACF and CCF of 10^7 fading variates generated by *Model II*. 120
4.25 LCR of 10^7 fading variates generated by *Model II*. 121
4.26 PDF of 10^7 fading variates generated by *Model II*. 121
4.27 CDF of 10^7 fading variates generated by *Model II*. 121
4.28 Layout of the 500 MHz semicustom fading channel variate generator. . . . 123

5.1 Direct-Form network structures. 129
5.2 (a) Direct-Form II. (b) Direct-Form I. 130
5.3 Cascade realization of the spectral shaping filter. 131
5.4 Architecture of a filter-based fading channel simulator. 136
5.5 The magnitude responses of elliptic filters composed of $K = 2, 3, 4, 5$ cascaded biquads with $f_D T_s = 0.2$. 137
5.6 Ideal low-pass filter specification. 138
5.7 Magnitude response of the elliptic shaping filter. 139
5.8 Zero/pole plot of the 32-bit quantized shaping filter. 139
5.9 Magnitude response of the first stage IIR low-pass interpolation filter. . . . 140

5.10 Phase response of the IIR low-pass filters for interpolation. 141
5.11 Magnitude response of the elliptic shaping filter. 142
5.12 Magnitude response of the first stage IIR low-pass interpolation filter. . . . 143
5.13 (a) The structure of cascading Gaussian variate generator and $N = K + P$ second-order sections. (b) Biquad datapath. 144
5.14 (a) Control data flow graph and (b) the datapath of the shaping filter and the first-stage low-pass interpolation filter. 145
5.15 (a) The structure of cascading Gaussian variate generator and the shaping elliptic IIR filter. (b) The IIR Chebyshev low-pass filter structure designed using cascading biquads. 146
5.16 Control data flow graph of shaping filter and first-stage interpolation. 146
5.17 ACF and CCF of 2.5×10^7 generated fading variates. 149
5.18 LCR of 2.5×10^7 generated fading variates. 149
5.19 PDF of 2.5×10^7 generated fading variates. 150
5.20 Symbol error rate for simulated 4-PSK and 16-QAM. 150
5.21 The architecture of the Python FP. 151
5.22 The datapath of biquad. 151
5.23 The microinstruction format. 152
5.24 ACF and CCF of 3×10^7 generated fading variates. 154
5.25 LCR of 3×10^7 generated fading variates. 154
5.26 PDF of 3×10^7 generated fading variates. 154

6.1 An (n_T, n_R) MIMO channel. 156
6.2 Sphere around the decoded point $\mathbf{p} = Q(\mathbf{H}^\dagger \mathbf{y})$. 166
6.3 LST transmitter architecture. 169
6.4 C•RAM processing element. 173
6.5 DSP-RAM architecture. 174
6.6 Architecture of one processing element. 175
6.7 Dataflow diagram of the symbol decoding process for one ST symbol. . . . 178
6.8 Dataflow diagram of the symbol decoding process for n_P ST symbols. . . . 178
6.9 Chip Plot of a 64-PE DSP-RAM in 0.18-μm CMOS. 182
6.10 Symbol error rate vs. SNR for ZF, MMSE and three different LST algorithms, for a (4,4) MIMO system utilizing 16-QAM modulation over a Rayleigh flat-fading channel. 182
6.11 SER vs. SNR for floating-point (FP) and four fixed-point number representations. 185

7.1 Logarithmic segmentation of the domain of $f(u_1)$. 192
7.2 Sub-segmentation of s_0. 192
7.3 MIMO channel simulator. 197
7.4 SER versus SNR for (4,4) MIMO system with a 4-QAM modulation. (b): The system is correlated in time. (a) The system is correlated in both time and space. 198
7.5 Effect of Training Length on the MSE of the Channel. 200

Nomenclature

List of Acronyms

ACC	Accumulator
ACF	Autocorrelation function
ADC	Analog-to-digital converter
AES	Advanced encryption standard
AOA	Angle of arrival
AR	Autoregressive
ARMA	Autoregressive moving average
ASIC	Application-specific integrated circuit
AWGN	Additive white Gaussian noise
BER	Bit error rate
BM	Box-Muller
bps	Bits per second
BW	Bandwidth
CAG	Cellular automata generator
CCF	Cross-correlation function
CDF	Cumulative distribution function
CDFG	Control data flow graph
CLB	Configurable logic block
CLT	Central Limit Theorem
CSCG	Circularly symmetric complex Gaussian
CSI	Channel state information

C•RAM	Computational RAM
dof	Degree of freedom
DPSK	Differential phase-shift keying
DSP	Digital signal processor
DTFT	Discrete-time Fourier transform
EICG	Explicit-inverse congruential generator
FFT	Fast Fourier transform
FIFO	First-in first-out
FIR	Finite-duration impulse response
FLOPS	Floating-point operations
FP	Filter processor
FPGA	Field-programmable gate array
FT	Fourier transform
GE	Gaussian elimination
GJ	Gauss-Jordan
GPP	General-purpose processor
GSM	Global system for mobile communications
GV	Gaussian variate
GVG	Gaussian variate generator
HDL	Hardware description language
HiperLAN	High performance radio local area network
i.i.d	Independent and identically distributed
ICG	Inverse congruential generator
IFFT	Inverse fast Fourier transform
IIR	Infinite-duration impulse response
ILP	Instruction-level parallelism
ILPF	Interpolation low-pass filter
ISA	Instruction set architecture

ISI	Inter-symbol interference
LCG	Linear congruential generator
LCR	Level crossing rate
LFG	Lagged Fibonacci generator
LFSR	Linear feedback shift register
LOD	Leading one detector
LOS	Line of sight
LST	Layered space-time
LTI	Linear time-invariant
LTV	Linear time-varying
LUD	LU decomposition
LUT	Look-up table
MA	Moving average
MAC	Multiply-accumulate
MC	Monte Carlo
MIMO	Multiple-input multiple-output
MISO	Multiple-input single-output
ML	Maximum likelihood
MLCG	Multiplicative linear congruential generator
MLCG	Multiplicative recursive linear congruential generator
MMSE	Minimum mean-squared error
MPS-MC	Multiple parameter set Monte Carlo
MRC	Maximal ratio combining
MSB	Most significant bit
MSE	Mean squared error
MSI	Multistream interference
MU	Mobile unit
OLSF	Orthogonal least squares fit

PDF	Probability density function
PE	Processing element
PHY	Physical layer
PIM	Processor-in-memory
PN	Pseudo-random number
PNG	Pseudo-random number generator
PSD	Power spectral density
QAM	Quadrature amplitude modulation
QoS	Quality of service
RHS	Right hand side
RISC	Reduced instruction set computer
rms	Root mean square
ROM	Read-only memory
RPP	Rapid prototyping platform
RTF	Rational transfer function
RV	Random variable
SD	Sphere decoding
SER	Symbol error rate
SIMD	Single-instruction multiple-data
SIMO	Single-input multiple-output
SISO	Single-input single-output
SNR	Signal-to-noise ratio
SOC	System-on-a-chip
SOR	Successive over-relaxation
SOS	Sum-of-sinusoids
sos	Second-order section
SRL	Shift register lookup table
ST	Space-time

STC	Space-time coding
STP	Space-time processing
SVD	Singular value decomposition
TDL	Tapped delay line
TG	Tausworthe generator
US	Uncorrelated scattering
VLIW	Very long instruction word
WSS	Wide-sense stationary
WSSUS	Wide-sense stationary uncorrelated scattering
ZF	Zero-forcing

List of Symbols

$*$	Convolution operator
$\mathbf{I}$	Identity matrix
$\dagger$	Moore-Penrose pseudo-inverse of a matrix
$\delta(\cdot)$	Kronecker delta function
$\Im\{x\}$	Imaginary component of x
λ	Wavelength
$\langle\cdot\rangle$	Inner product operator
$\mathbb{C}(\cdot)$	Set of complex numbers
$\mathbb{Q}$	Signal constellation
$\mathbb{R}(\cdot)$	Set of real numbers
$\mathbb{Z}(\cdot)$	Set of integer numbers
$\mathcal{CN}(m_X, \sigma_X^2)$	Circularly symmetric complex Gaussian distributed with mean m_X and variance σ_X^2
$\mathcal{J}_0(\cdot)$	Zeroth-order Bessel function of the first kind
$\mathcal{N}(m_X, \sigma_X^2)$	Normally distributed with mean m_X and variance σ_X^2
$\mathcal{O}(\cdot)$	Computational complexity
$\mathcal{Q}(\cdot)$	Symbol mapping function
$\mathcal{Q}\langle,\rangle$	Quantization format

$\mathcal{S}$	Signal space
$\mathcal{Z}[\cdot]$	z-transform
$\mathfrak{F}[\cdot]$	Fourier transform
$\text{erfc}(\cdot)$	Complementary error function
$\text{erf}(\cdot)$	Error function
$\text{E}[\cdot]$	Expected value operator
$\text{Pr}[\cdot]$	Probability
$\text{Tr}\{\cdot\}$	Trace of a matrix
$\text{Var}[\cdot]$	Variance
$\otimes$	Kronecker product
$\Re\{x\}$	Real component of x
f_c	Carrier frequency
f_D	Maximum Doppler frequency
N_o	Noise variance
n_R	Number of receiver antennas
n_T	Number of transmitter antennas
$Q(\cdot)$	Q function
$\mathbf{X}^*$	Conjugate transpose of X
$\mathbf{X}^H$	Hermitian of matrix $\mathbf{X}$
$\mathbf{X}^T$	Transpose of $\mathbf{X}$

Chapter 1

Motivation

Wireless communication systems have evolved rapidly to meet user demands for higher spectral efficiency, better quality of service (QoS), and higher energy efficiency. Since the bandwidth and power are scarce resources, the capacity limitations of single antenna systems make these systems unsuitable for many high data rate applications. Smart (i.e., adaptive) antenna systems have increased efficiency by using multiple antennas at one side of the communication link (typically the less power-constrained base-station side) with a single antenna on the client device [1]. Such systems utilize diversity schemes to mitigate multipath fading, use multiple channels to increase capacity, and use beamforming for interference reduction.

As the requirements to increase the data rate and QoS have continued to rise, and while limited bandwidth and power continue to pose severe limitations, multiple-input multiple-output (MIMO) systems have emerged as a new paradigm for wireless communications. MIMO systems use multiple transmitter *and* receiver antennas to improve the reliability and robustness of wireless communication links and to increase the data throughput in the presence of rich multi-path fading, without increasing the transmitted power or signal bandwidth [2]. While traditional wireless communication systems *mitigate* multipath propagation effects, MIMO is the first communication technique that *exploits* multipath propagation to increase link capacity. It has been shown that in a richly-scattered channel, such as in indoor wireless communications, for high enough signal-to-noise ratio (SNR) values, the spectral efficiency grows linearly with the smaller of the number of transmitter or receiver antennas [2]. This capacity increase can greatly exceed that of systems with a single antenna at one or both ends of the communication link.

The significant theoretical advantages of MIMO technology introduce several chal-

lenges to communication system designers. In addition to the size, cost, and complexity of the radio front end that scales with the number of antennas, the signal processing required to recover the transmitted symbols from the jumble of received signals at the MIMO receiver is computationally demanding [3]. For example, signal decoding using optimal algorithms increases the computational complexity exponentially in the number of antennas, which is prohibitive even for moderate number of antennas [4]. The high computational demands of MIMO signal decoding at the receiver can exceed the performance available from even high-end DSPs. Therefore, an important challenge is to investigate architecture and circuit techniques that facilitate implementation of the required computationally-intensive signal processing algorithms in a power-efficient and cost-effective manner.

Another important design challenge is the rapid evaluation of alternative signal processing algorithms in order to reduce design time and, hence, allow faster time to market. Many physical layer (PHY) algorithms have been proposed in the literature to meet key goals such as high data rate and low probability of error. Thus, hardware system designers face a large variety of alternative PHY algorithms, possibly with very different computational complexities, that must be evaluated in the early stages of the design cycle. Decisions made at the algorithm design phase are quite important because they have dramatic impact on the rest of the product development process. Hence, a significant part of the design effort must be dedicated to the simulation and verification of PHY signal processing algorithms. Moreover, the process of candidate algorithm selection and evaluation may require several time-consuming iterations.

While software simulations are widely used in the design and verification of wireless communication systems, the main drawback of conventional software-based Monte Carlo (MC) simulation is the increasingly long required simulation times. The simulation time is mainly related to the error rate performance [5] (the lower the error rate, the longer the simulation) and also to the channel conditions. For example, the slower the fading, the longer the fade duration, thus requiring longer simulation times to get meaningful results. Serial instruction execution and the lack of specialized hardware for MC simulation will also lengthen the simulation time. Hence, the simulation time can become unacceptably long especially when evaluating the performance of candidate algorithms that operate at very low error rates, over slow-fading wireless channels on a general-purpose processor (GPP).

When the MC simulation technique is used to evaluate the error rate performance of

communication systems, the statistical properties of the inputs and also the accuracy of the approximated models impact the simulation results. All wireless communication systems operate over a wireless channel with adverse propagation conditions such as time-varying multipath fading. Since the resulting error rate strongly depends on radio channel characteristics, it is important to use a real radio channel when designing and testing mobile communication systems. However, field testing to obtain empirical measurements is costly and inflexible, preventing the more thorough exploration of alternatives. Also, propagation conditions are almost impossible to repeat for the purpose of comparative analysis. Moreover, field testing is hard to generalize because different locations have different geometry structures. Another option is to use commercially available but costly and bulky fading channel emulators [6, 7]. Therefore, another challenge is to accurately model propagation characteristics for the simulation/prototyping platform. While there are various published models for wireless fading channels, a thorough analysis of these models is required to make sure that the chosen model reproduces the statistical properties of the real world channel as faithfully as possible [8].

Another challenge addressed in this thesis is the accurate modeling of noise at the receiver, which is commonly modeled as a sequence of variates with a Gaussian probability distribution function (PDF). Since small values of noise variates are readily tolerated by systems that operate at a very low error rate, random variates near the center of the distribution do not contribute significantly to the probability of error. For a MC simulation, the PDF of generated random numbers must be especially close to the true Gaussian PDF at the high σ regions (the tails of the PDF), where σ denotes the standard deviation of the Gaussian distribution. Since the tail of the Gaussian PDF decays exponentially, another important challenge in the rapid performance evaluation of communication systems is the fast generation of Gaussian variates (GVs) with accurate PDF, especially at the tails of distribution.

1.1 Thesis Contributions

This thesis makes contributions in four areas:

• We describe the design and implementation of the fastest and most compact disclosed digital Gaussian variate generator (GVG) with accurate statistical properties. The GVG occupies only 1% of a single Xilinx Virtex-II XC2V4000-6 Field programmable gate array (FPGA) and operates at 253 MHz [9], generating 506 million GVs per second within a

range of $\pm 9.41\sigma$. The design can be easily configured to achieve higher tail accuracy at a small cost in extra hardware and with only slightly decreased operating rate.

- Two compact implementations of a fading channel simulator based on the sum-of-sinusoids (SOS) approach are described. The implemented SOS-based fading simulator uses only 1% of the Xilinx Virtex2P XC2VP100-6 FPGA and operates at 221 MHz, generating 221 million complex fading coefficients per second.

- An improved SOS-based fading channel model is presented. The proposed model improves the statistical properties of generated fading variates compared to previously proposed models. A fixed-point implementation of the fading channel simulator on a Xilinx Virtex-II XC2V4000-6 FPGA utilizes only 5% of the configurable resources and generates over 200 million 16-bit fading variates per second.

- A much more compact and yet accurate implementation of a parameterized fading channel simulator using digital infinite-duration impulse response (IIR) filters is described. A novel filter design scheme is proposed to implement both the shaping filter and the interpolation low-pass filters together on a single FPGA. Conventional implementations are commonly realized on heterogeneous architectures (usually consisting of GPPs, DSPs, FPGAs, etc.) to implement the required computationally-intensive multi-rate signal processing algorithms of filter-based techniques. The new design is the first digital baseband fading channel simulator that is realizable on a fraction of a single FPGA. The fixed-point implementation of Rayleigh fading channel simulator on a Xilinx Virtex-II XC2V4000-6 FPGA utilizes only 4% of the configurable slices, 20% of the dedicated multipliers, and 2% of the available memories on a Xilinx Virtex2P XC2VP100-6 FPGA, while generating 25 million fading variates per second. The parameterized mobile channel simulator can be reconfigured to accurately simulate a wide variety of different channel characteristics.

- A flexible and compact filter processor architecture, called "Python", is designed to efficiently implement a multipath fading channel simulator on FPGAs. Python uses a simple and short instruction set to generate multiple sequences of fading variates for simulating wideband and MIMO channels. The Python filter processor uses only 2% of the configurable slices, 9% of dedicated multipliers and 14 on-chip BlockRAMs on a Xilinx Virtex-II XC2V4000-6 FPGA.

- An existing moderately-parallel and scalable architecture, called DSP-RAM, that combines the single-instruction multiple-data (SIMD) and *processor-in-memory* (PIM) approaches to increase the performance of moderately data-parallel signal processing applica-

tions is applied efficiently to the MIMO signal decoding problem. Integrating simple fixed-point datapaths, also called processing elements (PEs), with the local memories exposes the enormous data bandwidth between the two, and eliminates the bottleneck that otherwise occurs on an external bus between the memory chips and processor(s) in conventional architectures. The DSP-RAM can be readily mapped to standard FPGAs. By efficiently mapping the layered space-time (LST) MIMO algorithm onto the DSP-RAM architecture, it is shown that for a typical indoor wireless environment, a 100-MHz DSP-RAM can potentially provide more than 10 times greater decoding throughput at the receiver of a (4,4) MIMO system compared to a conventional 720-MHz DSP. The degree of parallelism (i.e., the number of PEs) can be easily scaled up to increase the throughput of a parallel algorithm. Also, one has the option of using increased parallelism to run at a slower clock frequency to simplify the implementation and still meet the required processing performance.

1.2 Thesis Outline

The thesis is organized as follows: Chapter 2 starts with a review of background material on random processes, linear systems and different transformations, and base-band signal processing. It briefly presents two fundamental components that are used to characterize wireless systems, multipath fading channels and noise models. Array antenna wireless systems and tradeoffs in the published transmission strategies are discussed next. The tradeoffs in algorithm efficiency and architecture for such components are explored. The feasibility of prototyping on FPGAs for rapidly evolving wireless standards is further discussed.

Chapter 3 presents the design and implementation of a fast, compact and accurate GVG. In this chapter, various candidate algorithms for generating GVs are compared. The statistical properties of different digital pseudo-random number generators are evaluated and their impact on the accuracy of generated GVs is discussed. Efficient implementations of trigonometric functions are considered. Various standard statistical tests are applied to the implemented GVG and the test results are presented.

Chapter 4 considers modeling and implementation of SOS-based Rayleigh fading channel simulators. In this chapter, various SOS-based fading channel models are presented and their statistical properties are compared. Two compact implementations of the most accurate SOS-based fading simulator are presented. Also, a novel fading channel model based on the SOS approach is presented. This model accurately reproduces the desired statistical properties of the standard Rayleigh fading envelope. Implementation results of the

proposed fading simulator on several widely available FPGAs are given.

Chapter 5 presents a novel design and implementation scheme to realize a parameterized fading channel simulator on a single FPGA. A new IIR filter design is presented to implement the required shaping filter and interpolation low-pass filters together to efficiently implement a compact fading channel simulator. Also, a flexible and compact filter processor architecture is presented that can simultaneously generate multiple independent sequences of fading variates for simulating wideband and MIMO channels.

Chapter 6 presents an efficient parallel algorithm and architecture for implementing LST decoding for MIMO systems. The computational complexity of different detection schemes, such as maximum likelihood, lattice decoders, and LST decoders, are compared. Efficient mappings of common MIMO detection algorithms are implemented and evaluated.

Conclusions and promising directions for future work are discussed in Chapter 7.

Chapter 2

Introduction

Typically, digital communication system designers have a set of goals to meet including minimizing the required system bandwidth, maximizing the transmission bit rate, minimizing the probability of bit error, minimizing the required power (transmit and computational), maximizing system utilization (i.e., to provide reliable QoS for a maximum number of users with minimum delay and maximum resistance to interference), and minimizing the system complexity and computational load. Other important practical and economic objectives include minimizing the time to market, the physical size, and the overall cost. In the available design space trade-offs, decisions must be carefully made as they might strongly impact other objectives. In this chapter, we will review architectural tradeoffs, efficiency measures for signal processing algorithms, and the significant role of hardware prototyping and hardware-accelerated characterization when developing wireless communication systems.

This chapter is organized as follows. Section 2.1 briefly reviews the required background information that is referenced throughout this thesis. Specifically, different frequency domain representations of signals, random processes and statistical properties, linear systems, multirate signal processing, baseband processing, digital modulation, and the geometric view of signals are presented. A standard noise model for the receiver in communication systems is discussed in Section 2.2. The impact of the wireless channel on the transmitted signals is briefly presented in Section 2.3. Section 2.4 reviews the throughput of different transmission strategies. Different measures of algorithm efficiency are discussed in Section 2.5. The architectural design space for wireless applications is presented in Section 2.6. Finally, the feasibility of rapid prototyping and its significance in the design cycle of wireless algorithms is discussed in Section 2.7.

2.1 Background

2.1.1 Frequency Domain Representations of Signals

Fourier analysis defines the frequency-domain representation of a given signal $x(t)$ in that it specifies the complex amplitude of the various frequency components (or spectral content) of the signal [10, 11]. A *Fourier series* allows a periodic signal to be decomposed into a sum of real-valued sine and cosine waveforms (or, more generally, a sum of complex exponentials). However, most signals are aperiodic. The *Fourier transform* (FT) is used to analyze the frequency content of an aperiodic signal. The FT of the signal $x(t)$ can be obtained using the *analysis equation*

$$X(f) = \int_{-\infty}^{\infty} x(t) \exp(-j2\pi ft) \mathrm{d}t$$

where $x(t)$ can be written using the *synthesis equation*

$$x(t) = \int_{-\infty}^{\infty} X(f) \exp(j2\pi ft) \mathrm{d}f.$$

In general, the FT $X(f)$ is a complex function of frequency f that may be expressed in the form

$$X(f) = |X(f)|e^{j\theta(f)}$$

where the amplitude function $|X(f)|$ is called the continuous *magnitude spectrum* of $x(t)$ and $\theta(f)$ is the continuous *phase spectrum* of $x(t)$. The result of computing a Fourier transform is sometimes referred to as the Fourier spectrum or simply the *spectrum*. The FT analysis and synthesis equation can also be written in terms of angular frequency ω as

$$\begin{aligned} X(\omega) &= \int_{-\infty}^{\infty} x(t) \exp(-j\omega t) \mathrm{d}t, \text{ and} \\ x(t) &= \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega) \exp(j\omega t) \mathrm{d}\omega, \end{aligned}$$

respectively.

Assume a signal $x(t)$ is sampled at intervals $T_s = 1/F_s$, where the k-th sample corresponds to $x(t = kT_s)$, and the last sample is at $k = K - 1$. If the signal is causal (i.e., the first sample is at $k = 0$), giving a total K samples, the *discrete-time Fourier transform* (DTFT) of $x(t)$ can be written as

$$X(f) = \sum_{k=0}^{K-1} x[k] \exp\left(\frac{-j2\pi nk}{K}\right). \tag{2.1}$$

By assuming a finite number of data points, we implicitly assume that $x(t)$ is periodic with a period of K samples (or KT_s seconds). Hence we need to evaluate the above equation at zero frequency and at harmonics of the *fundamental frequency* $f_o = 1/(KT_s) = F_s/K$ (i.e., K discrete frequencies $0, f_o, \cdots, (K-1)f_o$). We can write the DTFT of $x(t)$ by simplifying equation (2.1) using the time index k and the frequency index n as

$$X[n] = \sum_{k=0}^{K-1} x[k] \exp\left(\frac{-j2\pi nk}{K}\right), \quad n = 0, 1, , \cdots, K/2.$$

The reason that $n = 0, 1, , \cdots, K/2$ is that since the discrete signal $x[k]$ is sampled at F_s, then the signal has *image components* above $F_s/2$. In other words, the DTFT of a signal $x(t)$ is periodic in the frequency domain. When evaluating equation (2.1), it is only necessary to evaluate it up to $F_s/2$ (i.e., first $K/2 - 1$ discrete frequency samples). Thus we need only one period of $X(e^{j\omega})$ (i.e., $\omega \in [0, 2\pi]$ or $[-\pi, \pi]$, etc) for analysis and not the whole infinite domain. The inverse DFT can be written as

$$x[k] = \frac{1}{K} \sum_{n=0}^{K-1} X[n] \exp\left(\frac{j2\pi nk}{K}\right).$$

A fast technique to calculate the DTFT of a signal is the *fast Fourier transform* (FFT) algorithm that takes advantage of the fact that the calculation of the coefficients of the DTFT can be carried out in an iterative manner [11]. It is shown that to compute the DTFT of a sequence of K samples using the FFT algorithm, in general $K \log_2 K$ complex additions and $K \log_2 K$ complex multiplications are required (compared to direct implementation that requires $K(K-1)$ complex additions, K^2 complex multiplications) [12]. Hence, by using the FFT algorithm, the number of arithmetic operations is reduced by a factor of $K/\log_2 K$ which is considerable savings for relatively large K values.

2.1.2 Random Processes

Many random phenomena are functions of time. Consider a random experiment specified by the outcome ζ_k from some sample space $\Omega = \{\zeta_1, \zeta_2, \cdots\}$ with a probability $\Pr(\zeta_k)$. A function of time can be assigned to every outcome ζ_k that generates a sequence $X(t, \zeta)$. Hence, $X(t, \zeta)$ can be viewed as a function of two variables. When ζ has a fixed value ζ_k and n is treated as independent (non-random) index variable, the $X[n, \zeta_k]$ is called a *realization* or a *sample sequence* of the random process. We can think of $X[n, \zeta_k]$ as a vector of (possibly) infinite duration where the entire sequence is generated from a single

outcome of the underlying experiment. When ζ is treated as a variable and $n = n_k$ is fixed, $X[n_k, \zeta]$ is a *random variable* (RV). When both ζ and n are fixed, then $X[n_k, \zeta_k]$ is a number x. $X[n, \zeta]$ with both ζ and n treated as variables is called a *random process* (RP) or *stochastic process* [11]. A RP $X[n, \zeta]$ is thus a set of indexed RVs, one RV for each index variable n. If t is a continuous time variable then $X(t, \zeta)$ is a *continuous-time* random process, and if t is a discrete time index then $X[n, \zeta]$ is a *discrete-time* random process or a *random sequence*. The set of all possible sequences, $\{X[n, \zeta]\}$, constitutes an *ensemble* of sample sequences. The number of possible sample functions in such as ensemble is usually assumed to be extremely large; often it is infinite [11]. We sometimes suppress the ζ to simplify the notation and use $X[n]$ to denote both random sequences and single realizations.

2.1.2.1 Specifying Random Variables

We are typically interested in specifying the joint behaviour of random variables within a family (i.e., the stochastic process at various time instants). Here we will start with random variables and then we will present important statistical properties of random processes. A random variable $X(\zeta)$ represents the functional relationship between a random event ζ and a real number x. For notational convenience we denote $X(\zeta)$ by X. The distribution function $F_X(x) = \Pr(X \leq x)$ represents the probability that the value of random variable X is less than or equal to a real number x, and is called the *cumulative distribution function* (CDF). The *probability density function* (PDF) is defined as $f_X(x) = \mathrm{d}F_X(x)/\mathrm{d}x$. Thus the probability of an event X over the domain interval $[x_1, x_2]$ equals

$$\Pr(x_1 \leq X \leq x_2) = \Pr(X \leq x_2) - \Pr(X \leq x_1) = F_X(x_2) - F_X(x_1) = \int_{x_1}^{x_2} f_X(x)\mathrm{d}x.$$

In other words, the probability of the events $\{x_1 \leq X \leq x_2\}$ is the area under the PDF over the domain $x_1 \leq X \leq x_2$. The probability that X has a value x can be written as

$$\Pr(x \leq X \leq x + \Delta x) \approx f_X(x)\Delta x.$$

In the limit as Δx approaches zero, $\Pr(X = x) = f_X(x)$. The *mean* m_X or *expected value* $\mathrm{E}[X]$ of a random variable X is defined as

$$m_X = \mathrm{E}[X] = \int_{-\infty}^{\infty} x f_X(x)\mathrm{d}x \quad \text{or} \quad \mathrm{E}[X] = \sum_x x f_X(x)$$

for a continuous-valued and a discrete-valued random variable X, respectively, where $\mathrm{E}[\cdot]$ is the expected value operator. The *variance* of X is defined as

$$\sigma_X^2 = \int_{-\infty}^{\infty} (x - m_X)^2 f_X(x) \mathrm{d}x = \mathrm{E}\big[(X - m_X)^2\big] = \mathrm{E}[X^2] - m_X^2$$

where $\mathrm{E}[X^2]$ is called the *mean-square value* of X. Similarly, for complex-valued random variables, the variance is defined as

$$\sigma_X^2 = \mathrm{E}\big[|X|^2\big] - \big|\mathrm{E}[X]\big|^2 = \mathrm{E}[XX^*] - \mathrm{E}[X]\big(\mathrm{E}[X]\big)^*$$

where X^* denotes the complex conjugate of X.

Consider a pair of random variables X_1 and X_2. The joint moment $\mathrm{E}[X_1 X_2]$ is defined as their *correlation*. The correlation of *centered random variables* $X_1 - \mathrm{E}[X_1]$ and $X_2 - \mathrm{E}[X_2]$ is

$$\mathrm{E}\big[(X_1 - m_{X_1})(X_2 - m_{X_2})\big] = \mathrm{E}[X_1 X_2] - m_{X_1} m_{X_2}$$

and is called the *covariance* of X_1 and X_2. Two random variables are *uncorrelated* if their covariance is zero, which is equivalent to $\mathrm{E}[X_1 X_2] = \mathrm{E}[X_1]\mathrm{E}[X_2] = m_{X_1} m_{X_2}$. Statistical independence can be applied to random variables defined on a sample space generated by combined experiments or by repeated trials of a single experiment. If the experiments result in mutually exclusive outcomes, then the probability of an outcome in one experiment is independent of an outcome in any other experiment. Multidimensional random variables $X_1, \cdots, X_n$ are said to be *statistically independent* if and only if

$$f(x_1, x_2, \cdots, x_n) = f_{X_1}(x_1) f_{X_2}(x_2) \cdots f_{X_n}(x_n)$$

or equivalently

$$F(x_1, x_2, \cdots, x_n) = F_{X_1}(x_1) F_{X_2}(x_2) \cdots F_{X_n}(x_n).$$

If X_1 and X_2 are statistically independent, then they are also uncorrelated; however, if they are uncorrelated, they are not necessarily statistically independent. X_1 and X_2 are *orthogonal* when X_1 and X_2 are uncorrelated and either one or both of the random variables has zero mean (i.e., $\mathrm{E}[X_1 X_2] = 0$).

2.1.2.2 Specifying Random Processes

Similar to statistical averages for random variables, statistical averages can also be defined for stochastic processes. Such averages are called *ensemble averages*. A RP can be

specified by its joint probability on $X(t_1), \cdots, X(t_m)$ over all finite sets of time indexes. Commonly, a RP is described by its mean and autocorrelation functions. The *mean* $m_X(t_k)$ of stochastic process $X(t)$ is a function of time and defined as

$$m_X(t_k) = \mathrm{E}[X(t_k)] = \int_{-\infty}^{\infty} x f_{X(t_k)}(x) \mathrm{d}x$$

where $X(t_k)$ is a random variable obtained by observing the random process at time t_k and $f_{X(t_k)}(x)$ is the PDF of $X(t_k)$ (the density over the ensemble of outcomes at time t_k). The *autocorrelation* $\mathrm{R}_X(t_1, t_2)$ is a function of t_1 and t_2 and is defined as the joint moment of $X(t_1)$ and $X(t_2)$ (random variables obtained by observing $X(t)$ at times t_1 and t_2, respectively)

$$\mathrm{R}_X(t_1, t_2) = \mathrm{E}[X(t_1)X(t_2)]$$

where if $X(t)$ is a complex-valued random process

$$\mathrm{R}_X(t_1, t_2) = \mathrm{E}[X(t_1)X^*(t_2)].$$

The *auto-covariance* $\mathrm{K}_X(t_1, t_2)$ is defined as the covariance of $X(t_1)$ and $X(t_2)$ as

$$\mathrm{K}_X(t_1, t_2) = \mathrm{E}\Big[\big(X(t_1) - m_X(t_1)\big)\big(X(t_2) - m_X(t_2)\big)\Big].$$

If the process is *zero mean* (i.e., $\mathrm{E}[X(t)] = 0$ for each t) then $\mathrm{K}_X(t_1, t_2) = \mathrm{E}\big[X(t_1)X(t_2)\big]$. The variance of $X(t)$ can be obtained from $\mathrm{K}_X(t_1, t_2)$

$$\mathrm{Var}\big[X(t)\big] = K_X(t, t) = \mathrm{E}\Big[\big(X(t) - m_X(t_1)\big)^2\Big].$$

The *cross-covariance* $\mathrm{K}_{XY}(t_1, t_2)$ of $X(t)$ and $Y(t)$ is defined by

$$\begin{aligned} \mathrm{K}_{XY}(t_1, t_2) &= \mathrm{E}\Big[\big(X(t_1) - m_X(t_1)\big)\big(Y(t_2) - m_Y(t_2)\big)\Big] \\ &= \mathrm{R}_{XY}(t_1, t_2) - m_X(t_1)m_Y(t_2) \end{aligned}$$

where $\mathrm{R}_{XY}(t_1, t_2) = \mathrm{E}\big[X(t_1)Y(t_2)\big]$ is the cross-correlation. Similarly, the covariance of two real-valued random vectors $\mathbf{X}$ and $\mathbf{Y}$ can be defined using a *covariance matrix* as

$$\mathrm{K}_{\mathbf{XY}} = \mathrm{E}\Big[\big(X - \mathrm{E}[\mathbf{X}]\big)\big(\mathbf{Y} - \mathrm{E}[\mathbf{Y}]\big)\Big].$$

To specify the covariance matrix of two complex random vectors, four real-valued matrices are required

$$\begin{bmatrix} \mathrm{K}_{X_i Y_i} & \mathrm{K}_{X_i Y_q} \\ \mathrm{K}_{X_q Y_i} & \mathrm{K}_{X_q Y_q} \end{bmatrix}.$$

Two stochastic processes $X(t)$ and $Y(t)$ are *orthogonal* if $\mathrm{R}_{XY}(t_1,t_2) = 0$ for all t_1 and t_2. The processes $X(t)$ and $Y(t)$ are said to be *uncorrelated* if $\mathrm{K}_{XY}(t_1,t_2) = 0$ for all t_1 and t_2 (i.e., $\mathrm{R}_{XY}(t_1,t_2) = \mathrm{E}[X(t_1)]\mathrm{E}[Y(t_2)]$).

A random process $X(t)$ is a *Gaussian* RP if the random variables $X_1 = X(t_1), \cdots,$ $X_k = X(t_k)$ are jointly Gaussian random variables for all k and all choices of $t_1, \cdots, t_k$. If $X[n]$ is a sequence of independent Gaussian random variables with mean m_X and variance σ_X^2, then (1) the sum process has mean nm_X and variance $n\sigma_X^2$, and (2) the covariance matrix for $t_1, \cdots, t_k$ is $K_X(t_i,t_j) = \sigma_X^2\mathbf{I}$. A *zero mean Gaussian process* $X(t)$ is a zero mean RP for which, for any integer $m > 0$ the RVs $X(t_1), \cdots, X(t_m)$ are jointly Gaussian (and, of course, zero mean). For a RP $X = (X_1, X_2, \cdots, X_m)$, the covariance between each pair of RVs can be represented by the *covariance matrix* $\mathrm{K}_X = \mathrm{E}[XX^T]$. For a vector of normalized *independent and identically distributed* (i.i.d) Gaussian RVs, $\mathrm{E}[X_iX_j] = 0$ for $i \neq j$ and one for $i = j$. Thus $\mathrm{K}_X = \mathbf{I}_m$.

A *complex-valued random process* can be written as $Z(t) = X(t) + jY(t)$ where $X(t)$ and $Y(t)$ are random processes for the real and imaginary components, respectively. The complex-valued random process is commonly used in the representation of narrow-band band-pass signals and noise in terms of equivalent low-pass components. Important properties of $Z(t)$ can be expressed by its autocorrelation function (ACF) defined as

$$\begin{aligned}\mathrm{R}_{ZZ}(t_1,t_2) &= \frac{1}{2}\mathrm{E}[Z(t_1)Z^*(t_2)] = \frac{1}{2}\mathrm{E}\Big[\big[(X(t_1)+jY(t_1)\big]\big[(X(t_2)-jY(t_2)\big]\Big] \\ &= \frac{1}{2}\Big\{\mathrm{R}_{XX}(t_1,t_2)+\mathrm{R}_{YY}(t_1,t_2)+j\big[\mathrm{R}_{YX}(t_1,t_2)-\mathrm{R}_{XY}(t_1,t_2)\big]\Big\},\end{aligned}$$

where $\mathrm{R}_{ZZ}(t_1,t_2)$ sometimes denoted by $\mathrm{R}_Z(t_1,t_2)$.

2.1.2.3 Stationary Random Processes

Many random processes have the property that the nature of the randomness in the process does not change with time. In fact, an observation of the process within the time interval (t_1,t_2) exhibits the same statistical properties as an observation in some other time interval $(t_0+\tau, t_1+\tau)$. In this case, the random process is called a *stationary* RP. A RP is stationary in the *strict sense* if none of its statistics are affected by a shift in time. The mean (and variance) of a stationary RP is constant and independent of time as

$$m_X(t) = m_X. \tag{2.2}$$

Also, the auto-correlation and the auto-covariance of $X(t)$ can depend only on the time difference $t_2 - t_1$ where

$$\begin{aligned} \mathrm{R}_X(t_1, t_2) &= \mathrm{R}_X(t_2 - t_1) = \mathrm{R}_X(\tau) = \mathrm{E}\big[X(t)X(t+\tau)\big] = \mathrm{E}\big[X(t+\tau)X(t)\big];\ \forall t_1, t_2 \\ \mathrm{K}_X(t_1, t_2) &= \mathrm{K}_X(t_2 - t_1);\ \ \forall t_1, t_2. \end{aligned} \tag{2.3}$$

The conditions in Equation (2.2) and (2.3) are not sufficient to guarantee that $X(t)$ is strictly stationary. However, if these conditions hold, then $X(t)$ is *wide-sense stationary* (WSS) or *stationary in the wide sense* [11]. Typically, in communication systems random processes are considered complex WSS. Also, for practical applications, it is not necessary that a RP be stationary for all time, but only over some observation interval of interest. A complex RP is WSS if its real and imaginary parts are jointly WSS. For a complex-valued WSS random process,

$$\mathrm{R}_X(\tau) = \mathrm{E}\big[X(t+\tau)X^*(t)\big]$$

where $\mathrm{R}_X(0) = \mathrm{E}\Big[\big|X(t)\big|^2\Big]$ is the *second moment* of the samples. If the WSS RP is discrete then

$$\mathrm{R}_X[m] = \mathrm{E}\big[X[k+m]X^*[k]\big]$$

where $m = k - i$ and $\mathrm{R}_X[0] = \mathrm{E}\big[|X[k]|^2\big]$ can be interpreted as the *power* of the random process.

Consider two random processes $X_1(t)$ and $X_2(t)$ with ACF $\mathrm{R}_{X_1}(t_1, t_2)$ and $\mathrm{R}_{X_2}(t_1, t_2)$, respectively. The two *cross-correlation functions* (CCFs) of $X_1(t)$ and $X_2(t)$ may be defined as

$$\begin{aligned} \mathrm{R}_{X_1,X_2}(t_1, t_2) &= \mathrm{E}\big[X_1(t)X_2(t)\big] \\ \mathrm{R}_{X_2,X_1}(t_1, t_2) &= \mathrm{E}\big[X_2(t)X_1(t)\big]. \end{aligned}$$

The correlation properties of the two RPs can be expressed as a *correlation matrix* as follows:

$$\mathrm{R}(t_1, t_2) = \begin{bmatrix} \mathrm{R}_{X_1}(t_1, t_2) & \mathrm{R}_{X_1X_2}(t_1, t_2) \\ \mathrm{R}_{X_2X_1}(t_1, t_2) & \mathrm{R}_{X_2}(t_1, t_2) \end{bmatrix}.$$

2.1.2.4 Power Spectral Density

A stationary RP is an infinite energy signal and thus its FT does not exist. The power spectrum of a random signal is obtained by computing the FT of the ACF, that is the distribution

of power with frequency. If $X(t)$ is a WSS RP with mean m_X and ACF $\mathrm{R}_X(\tau)$, then the power spectral density of $X(t)$ is given as

$$\mathrm{G}_X(f) = \mathfrak{F}\{\mathrm{R}_X(\tau)\} = \int_{-\infty}^{\infty} \mathrm{R}_X(\tau)\, e^{-j2\pi f\tau} \mathrm{d}\tau$$

where $\mathfrak{F}[\cdot]$ denotes the Fourier transform operation. Similarly, for a discrete-time WSS random process with mean m_X and ACF $\mathrm{R}_X[k]$, the PSD can be written as

$$\mathrm{G}_X[f] = \mathfrak{F}\{\mathrm{R}_X[k]\} = \sum_{k=-\infty}^{\infty} \mathrm{R}_X[k] e^{-j2\pi fk}.$$

$\mathrm{G}_X[f]$ is periodic in f with period one and hence we need only consider frequencies in the range of $-1/2 \leq f \leq 1/2$. In other words, $\mathrm{G}_X[f] = \mathrm{G}_X[f+k]$ for $k = \pm 1, \pm 2, \cdots$. This is a characteristic of the Fourier transform of any discrete-time sequence such as $\mathrm{R}_X[n]$ [11].

2.1.3 Linear Systems Response to Random Signals

Signal processing typically involves transformations from a time function into one or more other functions. If $x(t)$ is the input to a linear system and $y(t)$ is the system output,

$$y(t) = h(t) * x(t) = \int_{-\infty}^{\infty} h(\tau; t)\, x(\tau) \mathrm{d}\tau$$

where the operator $*$ denotes the convolution (which has commutative, associative, and distributive properties) and $h(\tau; t)$ is the system's impulse response. The system is real if its impulse response is real-valued and complex if its impulse response is complex-valued. The response of a *linear time-invariant* (LTI) system (i.e., holds additive and scaling properties) to an arbitrary input $x(t)$ is

$$y(t) = h(t) * x(t) = \int_{-\infty}^{\infty} h(\tau)\, x(t-\tau)\, \mathrm{d}\tau = \int_{-\infty}^{\infty} h(t-\tau)\, x(\tau)\, \mathrm{d}\tau \qquad (2.4)$$

where $h(t)$ is the impulse response of the LTI system, τ is the excitation time and t is the response time. $h(t-\tau)$ can be thought of as being $h(\tau)$ *folded* in time and *delayed* by t. The relation (2.4) is called the *convolutional integral* and it shows that an LTI system is completely specified by its impulse response. A system is called *stable* if every bounded (finite) input produces a finite output. For LTI systems, a necessary and sufficient condition for stability is that the impulse response must be absolutely integrable $\sum_{n=-\infty}^{\infty} |h[n]| < \infty$ [10]. A system is called *causal* if the output for $n = n_0$ depends only on the values of the input for $n \leq n_0$. For LTI systems, this implies that the impulse response sequence is zero

for $n < 0$. Thus the lower limit of integration in (2.4) can be changed to zero if the system is causal.

If $x[n]$ is a discrete-time signal that results from sampling a continuous-time signal, then $x[n]$ can be written as

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\,\delta[n-k]$$

where $\delta[n]$ denotes the Kronecker delta function in discrete time. Output $y[n]$ can thus be expressed as

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]\,h[n-k] = \sum_{k=-\infty}^{\infty} x[n-k]\,h[k].$$

If we assume that the input is a sequence $x[n] = e^{j\omega n}$ of complex exponentials, then

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]\,e^{j\omega(n-k)} = e^{j\omega n}\Big(\sum_{k=-\infty}^{\infty} h[k]\,e^{-j\omega k}\Big) = e^{j\omega n}H(e^{j\omega}).$$

Thus, the complex exponential sequence $e^{j\omega n}$ is an *eigenfunction* of LTI systems where the output response to sinusoidal input is sinusoidal with the same frequency as the input and with an amplitude and phase determined by the system (i.e., $H(e^{j\omega})$). The *eigenvalue* $H(e^{j\omega})$ is called the *frequency response* or the *transfer function* of the system. We can see that frequency response of an LTI system is simply the FT of the impulse response as follows

$$h[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H(e^{j\omega})e^{j\omega n}\mathrm{d}\omega.$$

Since $H(e^{j\omega})$ is a periodic function of the frequency with period 2π, we need only specify $H(e^{j\omega})$ over an interval of length 2π. In general $H(e^{j\omega})$ is complex and can be represented in polar form (i.e., in terms of magnitude and phase) as

$$H(e^{j\omega}) = |H(e^{j\omega})|e^{j\angle H(e^{j\omega})}.$$

Since the convolution of a pair of time functions is transformed into the multiplication of their Fourier transforms, we can write

$$Y(e^{j\omega}) = H(e^{j\omega})X(e^{j\omega}) \tag{2.5}$$

where the magnitude and phase of the FTs of the system input and output are related by

$$|Y(e^{j\omega})| = |H(e^{j\omega})|.|X(e^{j\omega})| \tag{2.6}$$

$$\angle Y(e^{j\omega}) = \angle H(e^{j\omega}) + \angle X(e^{j\omega}). \tag{2.7}$$

nonumber (2.8)

Here $|H(e^{j\omega})|$ is the *magnitude response* or *amplitude response* and $\angle H(e^{j\omega})$ is the *phase response* or the *phase shift* of the LTI system. As given in Equation (2.6), if $|H(e^{j\omega})|$ is small for a particular range of frequencies, then the frequency components of the input are suppressed in the output for those frequencies.

An alternative representation of a discrete-time linear system can be expressed using the *z-transform* of $h[n]$ as

$$H[z] = \mathcal{Z}[h[n]] = \sum_{n=-\infty}^{\infty} h[n]z^{-n}$$

where $z = |z|e^{j\omega}$ is a complex variable, $|z|$ is the attenuation and ω is the real angular frequency. Multiplication of $h[n]$ by z^{-n} corresponds to delaying the input sequence by n samples. The function $|z| = 1$ (or $z = e^{j\omega} = e^{j2\pi f} = e^{j\theta}$, if the unit of frequency is in radians/sec, hertz, or radians/samples, respectively) denotes a circle of unit radius in the complex z-plane and is called the *unit circle*. The complex function $H[z]$ is called the *system function* or *transfer function*. The values of $H[z]$ when evaluated on the unit circle in the z-plane give the *frequency response*. Similarly, it can be shown that [10] the z-transform of the output of an LTI system is related to the z-transform of the input and the z-transform of the system impulse response as

$$Y(z) = H(z)X(z)$$

where any LTI system is completely characterized by its system function $H(z)$.

An important class of LTI systems are the *ideal frequency-selective* filters (systems) where the frequency response is unity over a certain frequency ranges (i.e., $|H(e^{j\omega})| = 1$) and zero elsewhere. This implies that the filter passes complex exponentials at one set of frequencies and completely rejects the complex exponential at other frequencies. Filters usually are described in the time domain by their impulse response $h(t)$, or in the frequency domain by their magnitude frequency response $|H(\omega)|$. $h(t)$ is usually derived from the filter's frequency domain description rather than directly in the time domain and is usually expressed in complex low-pass equivalent form (explained below).

Among four common types of filters (i.e., low-pass, high-pass, band-pass and band-stop), the low-pass filter (LPF) has been used the most since the transfer function of other filters can be computed from the normalized low-pass filter through a standard transformation of variables [13]. The LPF selects the low-frequency components of the signal and rejects the high-frequency components. An ideal LPF with cut-off frequency ω_c can be

defined as a discrete-time LTI system with the frequency response

$$|H(e^{j\omega})| = \begin{cases} 1 & \text{for} \quad |\omega| < \omega_c \\ 0 & \text{for} \quad \omega_c < |\omega| \leq \pi \end{cases}$$

that is periodic with period 2π. Note that since the frequency response of discrete-time sequence is periodic, it is completely specified by its behaviour over the domain $-\pi < \omega < -\pi$. The corresponding impulse response is given by

$$h[n] = \frac{1}{2\pi} \int_{-\omega_c}^{\omega_c} e^{j\omega n} \, \mathrm{d}\omega = \frac{\sin \omega_c n}{\pi n}, \quad -\infty < n < \infty$$

which implies that the ideal LPF is non-causal ($h[n] \neq 0$ for $n < 0$) and also is not absolutely summable. The sequence $h[n]$ approaches zero as n approaches infinity (implying that $H(e^{j\omega})$ has discontinuity at $\omega = \omega_c$). Moreover, the phase response is zero.

Since the impulse response of an ideal LPF extends from $-\infty$ to ∞, the output of an ideal LPF cannot be completed with finite computation. A class of practical LTI systems that can be implemented as an approximation to ideal frequency-selective filters corresponds to the constant-coefficient difference equation. This class is further explained and utilized in Chapter 5.

2.1.4 Baseband Processing

Two particular sequences are extremely important in analyzing digital communication systems: sinusoidal sequences and complex exponential sequences. A *sinusoidal* sequence has the general form of

$$x[n] = A\cos(\omega_0 n + \phi), \; n \in \mathbb{Z}$$

where ω_0 is the frequency of the sinusoid and ϕ is the phase. For a periodic sequence with period $N \in \mathbb{Z}$

$$A\cos(\omega_0 n + \phi) = A\cos(\omega_0 n + \omega_0 N + \phi)$$

which is true for $\omega_0 N = 2\pi k$. A *complex exponential* sequence can be written as

$$x[n] = |A|e^{j(\omega_0 n + \phi)} = |A|\cos(\omega_0 n + \phi) + j|A|\sin(\omega_0 n + \phi)$$

where the real and imaginary parts of $\exp\left[j(\omega_0 n + \phi)\right]$ vary sinusoidally with n. For any complex exponential sequence of period N

$$e^{j\omega_0(n+N)} = e^{j\omega_0 n}$$

is true for $\omega_0 N = 2\pi k$. Two important points regarding these signals are: (1) Depending on the value of ω_0, the complex exponential or sinusoid sequence may not be periodic at all (e.g., if $\omega_0 = 1$ there are no integer values for N or k that satisfy the periodicity). (2) For a discrete-time sinusoid signal as ω_0 increases from 0 to π, $x[n]$ oscillates more and more rapidly, and as ω_0 increases further from π to 2π, the oscillations becomes slower. Thus, for complex exponential and sinusoid signals, values of ω_0 in the vicinity of $\omega_0 = 2\pi k$ for $k \in \mathbb{Z}$ are referred to as *low frequencies* (relatively slow oscillations), while values of ω_0 in the vicinity of $\omega_0 = (\pi + 2\pi k)$ are referred to as *high frequencies*.

2.1.4.1 Base-band Signals Representation

Consider a real-valued signal $s(t)$. The signal $s(t)$ is a *band-pass* signal if its FT $S(f)$ is non-negligible only in a band of frequencies over total bandwidth $2W$ centered about some carrier frequency f_c. In the majority of communication systems the bandwidth $2W$ is much smaller compared to f_c, and thus such a signal is referred to as a *narrow-band* signal. A band-pass waveform is also called a digital waveform (although the waveform is sinusoidal and is analog) because it is encoded with digital information. The band-pass signal (sometimes is called the carrier) $s(t)$ can be expressed as

$$s(t) = a(t)\cos\theta(t) = a(t)\cos\left[2\pi f_c t + \phi(t)\right]$$

where $a(t)$ is the time-varying amplitude called the (natural) *envelope* of the band-pass signal $s(t)$ and $\theta(t)$ is the time-varying angle. $\theta(t)$ is typically denoted as $\theta(t) = 2\pi f_c t + \phi(t)$ where $\omega_c = 2\pi f_c$ is the radian frequency of the carrier and $\phi(t)$ is the time-varying phase of the signal.

The band-pass signal $s(t)$ also can be written as

$$s(t) = \Re\{s_l(t)\exp(j2\pi f_c t)\} \tag{2.9}$$

where $s_l(t)$ is the *complex envelope* of the signal (i.e., base-band message or data in complex form) and $e^{j2\pi f_c t}$ is the *carrier* in complex form. The process of multiplying these two signals is called *modulation* and $s(t)$ (i.e., the real part of the product) is the transmitted signal. The spectrum of the complex envelope $s_l(t)$ is limited to the band $-W \leq f \leq W$ (typically less than a few MHz) and centered at the origin [12]. Thus $s_l(t)$ is a *low-pass* signal. During modulation the base-band waveform $s_l(t)$ is *frequency translated* by a carrier wave to a frequency that is much larger than the spectral content of $s_l(t)$.

In general, $s_l(t)$ is a complex-valued signal that can be expressed as

$$s_l(t) = s_{li}(t) + j s_{lq}(t) = |s_l(t)| e^{j\theta(t)} \tag{2.10}$$

where $|s_l(t)| = \sqrt{s_{li}^2(t) + s_{lq}^2(t)}$ and $\theta(t) = \tan^{-1} s_{lq}(t)/s_{li}(t)$ are both real-valued low-pass functions. Note that both $s_{li}(t)$ and $s_{li}(t)$ are limited to the band $-W \leq f \leq W$. A band-pass signal $s(t)$ in (2.9) can be produced by a *quadrature type modulator* as [11]

$$s(t) = s_{li}(t)\cos(2\pi f_c t) - s_{lq}(t)\sin(2\pi f_c t).$$

Whether we represent the band-pass signal $s(t)$ in terms of its in-phase and quadrature components, or in terms of its envelope and phase as in (2.10), the information content of $s(t)$ is completely represented by the complex envelope $s_l(t)$.

Let the signal $s(t)$ be applied to an LTI band-pass system with impulse response $h(t)$ and transfer function $H(f)$. The output also is a band-pass signal $y(t) = h(t) * s(t)$ [11]. The analysis of a band-pass system, which is complicated by the presence of the multiplicative factor $\exp(j2\pi f_c t)$, can be replaced by an equivalent (but simpler) low-pass analysis. According to the equivalence theorem [12], performing band-pass linear signal processing followed by frequency translation down to the base-band yields the same results as first converting the band-pass signal to base-band, then performing linear signal processing to the base-band signal. The complex envelope $y_l(t)$ of the output signal of a band-pass system can be obtained by convolving the complex impulse response $h_l(t)$ of the system with the complex envelope $s_l(t)$ of the input band-pass signal and can be written as

$$y_l(t) = y_i(t) + j y_q(t) = \left[h_{li}(t) + j h_{lq}(t)\right] * \left[s_{li}(t) + j s_{lq}(t)\right] \tag{2.11}$$

Using the distributive property of convolution in Equation (2.11), the in-phase and quadrature components of the complex envelope $y_l(t)$ can be written as

$$\begin{aligned} y_i(t) &= h_{li}(t) * s_{li}(t) - h_{lq}(t) * s_{lq}(t) \\ y_q(t) &= h_{lq}(t) * s_{li}(t) + h_{li}(t) * s_{lq}(t). \end{aligned}$$

Thus evaluating the response of a band-pass system to an input band-pass signal requires four convolution operations and two addition in the low-pass equivalent model. The final output $y(t)$ can be written as $y(t) = \Re\left[y_l(t)\exp(j2\pi f_c t)\right]$.

2.1.4.2 Modulation

A (message) symbol m_i is a group of q bits drawn from a finite symbol set or *constellation* $\mathbb{Q}$ where the size M of alphabet is 2^q. Each signal in an M-ary constellation (alphabet) can be related to a unique sequence of q bits. For a base-band transmission, each m_i will be represented by one of the *base-band pulse waveforms* $g_i(t)$, where $i = 1, \cdots, M$. Thus we will use the terms symbol and waveform interchangeably. For typical band-pass transmission, each $g_i(t)$ pulse will be represented by one of the band-pass waveforms $s_i(t)$. Hence, first the incoming binary data is mapped into complex symbols, then the sequence of complex symbols are mapped into base-band waveforms, and finally, base-band waveforms are modulated to pass-band signals.

Band-pass modulation is the process by which an information signal (digital symbol with duration T_s) is converted to a sinusoidal waveform. Translation (or shifting) of the spectrum of a low-pass or base-band signal $s_l(t)$ to a higher frequency involves multiplying the base-band signal with a carrier waveform $\cos(2\pi f_c t)$. This operation can be explained by the following two important properties of the Fourier transform

$$\begin{aligned} s(t - t_0) &\rightleftharpoons S(f)\exp(-j2\pi f t_0) \\ s(t)\exp(j2\pi f t_0) &\rightleftharpoons S(f - f_c). \end{aligned} \tag{2.12}$$

Thus if a signal $s(t)$ is time-shifted by t_0, the amplitude of $S(f)$ is unaffected but its phase is changed by $-2\pi f_c t_0$. Conversely, multiplication of $s(t)$ by a complex sinusoidal factor $\exp(j2\pi f t_0)$ is equivalent to frequency-shifting its FT in the positive direction by f_c.

Since each modulated sinusoid $s(t) = a(t)\cos[\omega_c t + \phi(t)]$ can be distinguished from other sinusoids with three available parameters (i.e., amplitude, frequency, and phase), different modulation techniques have been introduced [11]. For example, *phase shift keying* (PSK) modulation scheme uses

$$s_i(t) = \sqrt{\frac{2E_s}{T_s}}\cos[\omega_c t + \phi_i(t)], \quad 0 \le t \le T_s, \quad i = 1, \cdots, M$$

where $T_s = \log_2 M.T_b$ is the *symbol duration*, T_b is the time duration of each data bit, E_s is the *symbol energy* and the phase term has M discrete values $\phi_i = 2\pi i/M$. Clearly, binary PSK (BPSK) uses $M = 2$ and shifts the phase of waveform $s_i(t)$ to one of two states zero or π. In PSK modulation scheme, the signal constellation is chosen such that the amplitude is the same for all signal points, by placing the signal points on a circle in the complex signal space. In this scenario the transmitted information is carried by the phase of the carrier. In

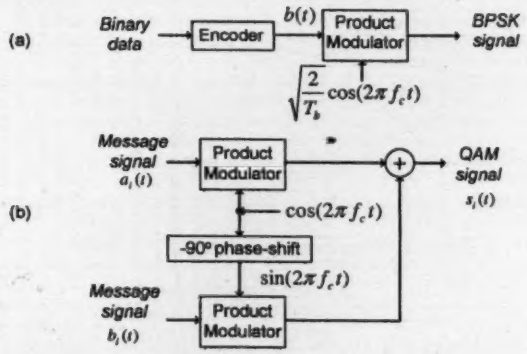

Figure 2.1: (a) BPSK modulator. (b) QAM modulator.

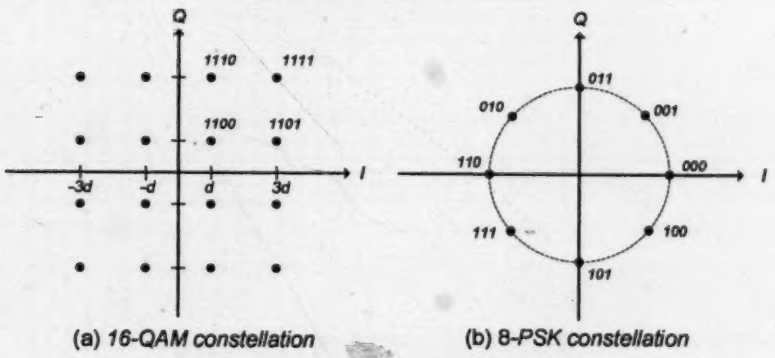

Figure 2.2: Two different constellations.

the more general case such as *quadrature amplitude modulation* (QAM), both amplitude and phase are allowed to change between signal alternatives [11]. A QAM signal can be expressed as

$$s_i(t) = a_i(t)\cos(\omega_0 t) + b_i(t)\sin(\omega_0 t)$$

where $[a_i(t), b_i(t)]$ denotes the i-th signal point in the QAM constellation. The block diagram of BPSK and QAM modulation are shown in Figure 2.1.

One common constellation is to place the signal points on a regular rectangular grid in the signal space. For example, in a 16-QAM modulation scheme, the signal points are spaced with the distance $2d$ along the axes, as shown in Figure 2.2(a). Assume each complex signal point $\mathbf{s}_k$ is represented as $\mathbf{s}_k = s_i + js_q$. When M is an even square (e.g., 16), there will be $\sqrt{M}$ possible amplitudes for both s_i and s_q. In fact, $s_i \in \{\pm d, \pm 3d, \cdots, \pm(\sqrt{M}-1)d\}$ and $s_q \in \{\pm d, \pm 3d, \cdots, \pm(\sqrt{M}-1)d\}$. While the signal power is proportional to the distance from signal point in the constellation to the origin,

the probability of error is related to the distance between the signal points in the constellation [12]. The energy of the signal $s_k(t)$ is $E_k = \|s_k(t)\|^2 = \|\mathbf{s}_k\|^2$ and the distance between $s_k(t)$ and $s_m(t)$ is $\|s_k(t) - s_m(t)\| = \|\mathbf{s}_k - \mathbf{s}_m\|$. Clearly, as M increases, the energy must be increased to allow the minimum distance between points to remain at the same value (i.e., to maintain a fixed error rate).

To transmit the symbol m_i, the digital waveform $s_i(t)$ will be transmitted over T_s seconds. The *data* (bit) *rate* can be written as $R = \log_2 M/T_s$ bits per second (bps). Similarly, the *symbol rate* can be written as $R_s = R/\log_2 M$. Thus the modulator yields an output symbol rate R_s that is a factor of $\log_2 M$ smaller than the input data bit rate R. The *bandwidth efficiency* of a digital communication system that transmits $\log_2 M$ bits in T_s second using a bandwidth of W Hz can be expressed as

$$\frac{R}{W} = \frac{\log_2 M}{WT_s} = \frac{1}{WT_b} \text{ bps/Hz.} \tag{2.13}$$

Note that in practice the occupied bandwidth is $W = R_s(1 + \alpha)$ where α is *excess bandwidth factor* [10]. The α factor is due to the fact that the shaping filter uses a raised cosine filter where its sharpness is described by the α. If the filter has a perfect ("brick wall") characteristic with sharp transitions (i.e., $\alpha = 0$), the occupied bandwidth would be equal to symbol rate.

2.1.4.3 Multirate Signal Processing

A real signal $x(t)$ is *band-limited* if there exists a finite frequency f such that $X(j2\pi f)$ is zero for $f > W$ where W is called the *signal bandwidth*. A band-limited signal $x(t)$ with bandwidth W can be reconstructed from its samples values $x[n] = x[nT_s]$ if the sampling frequency $F_s = 1/T_s$ is greater than twice the bandwidth W of $x(t)$, i.e., $F_s > 2W$ (or $T_s \leq 1/2W$) [10]. The sampling rate $2W$ for a band-limited signal is called the *Nyquist rate*. It is shown in [10] that the signal $x(t)$ can be reconstructed from its discrete-time samples as follows

$$x(t) = \sum_{n=-\infty}^{\infty} x[nT_s] \operatorname{sinc}\Big(\frac{t - nT_s}{T_s}\Big). \tag{2.14}$$

For a pass-band signal, the bandwidth W is defined as the bandwidth of the positive frequency part only (i.e., $-W/2 + f_c \leq f \leq W/2 + f_c$) and negative frequencies are not counted. The frequency of the equivalent base-band signal is $-W/2 \leq f \leq W/2$, which implies that the bandwidth is $W/2$. Thus the complex base-band signal has half of the bandwidth of the corresponding pass-band signal. If W is the double sided bandwidth of

the base-band signal $s_l(t)$ (i.e., the spectrum is non-zero only for $-W_n \leq W \leq W_p$, where $W = W_n + W_p$), then the sampling frequency of the quadrature branches of base-band signal each should be at least $F_s \geq W$. This should not be confusing as some interpretations use the highest frequency component as W (so if the lowest signal component is $-W$, then double sided bandwidth is $2W$) and the sampling frequency must be $F_s \geq 2W$.

According to (2.5), if $x(t)$ is bandlimited to W_1 and $h(t)$ is bandlimited to W_2, then $y(t)$ can be reconstructed without error if $F_s \geq 2W$ where $W = \max(W_1, W_2)$. If a system has several signals (or processes) with different bandwidths $W_1, \cdots, W_n$, then sampling at a single rate $F_s = 2W$, where $W = \max(W_1, \cdots, W_n)$, may entail unnecessary computation when the W_i's are relatively different. In addition to being inefficient, using a single sampling rate may introduce additional round-off errors due to the redundant computations. A more efficient processing scheme is for each random process (signal) to be sampled at a rate just sufficiently fast to satisfy the sampling theorem for that signal.

One feature of multi-rate digital signal processing systems is that the sampling rate of individual signals may have to be increased or decreased, leading to the two fundamental operations of *interpolation* (or up-sampling) and *decimation* (or down-sampling), respectively. If the original sampling rate is F_s (i.e., the input signal is bandlimited to half sampling rate), then to increase the sampling rate by an integer factor I, the output sequence should first be up-sampled by inserting I zero samples between each input sample. As a consequence, all the spectral images within $[-IF_s/2, IF_s/2]$ of the input spectrum appear at the output signal at the multiples of the input sample rate F_s. Then an interpolation low-pass filter (ILPF) is typically used to retain only the desired spectral components. The ILPF operates at a higher sampling rate than F_s, thus a flexible and yet computationally efficient scheme is desired. A polyphase structure is an efficient scheme that uses I parallel branches operating at the lower sampling rate [10]. For a sampling rate conversion by any rational factor I/D, the signal is first interpolated by I and then decimated by an integer factor D.

2.1.5 Geometric View of Signals and Transformations

A geometric (vector) view of signals is useful for representing base-band signals. A *signal space* $\mathcal{S}$ is a vector space that consists of a set of vectors that represent waveforms. Each waveform is represented by a *finite-energy complex function* $s(t)$. The function $s(t)$ can be

represented as a vector $\mathbf{s}$ where

$$||\mathbf{s}||^2 = \int_{-\infty}^{\infty} |s(t)|^2 \mathrm{d}t$$

is the energy in $\mathbf{s}$. A set of vectors, $\mathbf{s}_1, \cdots, \mathbf{s}_n$ in a vector space $\mathcal{S}$ is *linearly dependent* if $\sum_{i=1}^{n} \alpha_i \mathbf{s}_i = 0$ for some set of scalars α_i that are not all equal to 0. Equivalently, the sets is linearly dependent if any one of those vectors is a linear combination of others. A set of vectors is *linearly independent* if $\sum_{i=1}^{n} \alpha_i \mathbf{s}_i = 0$ only if each $\alpha_i = 0$. The *dimension* of a vector space is the number of vectors in the largest linearly independent set in that vector space. The *inner product* of two vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{N} x_i y_i^*. \tag{2.15}$$

Two vectors $\mathbf{x}$ and $\mathbf{y}$ are *orthogonal* if and only if $\langle \mathbf{x}, \mathbf{x} \rangle = 0$. Clearly, $\langle \mathbf{x}, \mathbf{x} \rangle = \sum_{i=1}^{N} |x_i|^2 = ||\mathbf{x}||_2$ where $||\mathbf{x}||$ is the *norm* of the vector $\mathbf{x}$ (i.e., the distance from $\mathbf{0}$ to $\mathbf{x}$ or the length of $\mathbf{x}$). Since the length or norm of a signal is the square-root of the signal energy, the energy of the difference of two signals can be interpreted as the square of the distance between the two signal vectors. When the receiver is aware of the set of possible signal waveforms, it uses the *minimum distance criterion* to choose the signal from the set of known symbols that is closest to the received signal as

$$\min \left\{ \|\mathbf{y} - \mathbf{s}_m\|^2 \right\} = \min \left\{ \|\mathbf{y}\|^2 + \|\mathbf{s}_m\|^2 - 2\Re\{\langle \mathbf{y}, \mathbf{s}_m \rangle\} \right\} \tag{2.16}$$

where $\mathbf{y}$ is the received signal (vector), $\mathbf{s}_m$ is the m-th signal chosen from a set of known signals $\{\mathbf{s}_m, 1 \leq m < M\}$.

Consider the linear transformation $\mathbf{Hs} = \mathbf{y}$. For every linear transformation, there is one and only one corresponding matrix. This transformation can be represented by the matrix-vector notation $\mathbf{Hs}$ or by a system of equations. The *transpose* of $\mathbf{H}$, denoted as $\mathbf{H}^T$, is the matrix obtained from $\mathbf{H}$ when the rows and columns are exchanged. One important property of the transpose operator is that $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$. A square matrix $\mathbf{H} \in \mathbb{R}^{n \times n}$ is *orthogonal* if $\mathbf{H}\mathbf{H}^T = \mathbf{I}$, where $\mathbf{I}$ is an identity matrix. Thus an orthogonal matrix is always invertible. If $\mathbf{H}$ is orthogonal, then its columns $\mathbf{h}_1, \cdots, \mathbf{h}_n$ (or its rows) form an *orthonormal basis* for $\mathbb{R}^n$ (i.e., $|\mathbf{h}_i| = 1$ and $\langle \mathbf{h}_i, \mathbf{h}_j \rangle = 0$ for all $i \neq j$). If there is a vector $\mathbf{v}\mathbb{R}^n \neq 0$ such that $\mathbf{Hv} = \sigma\mathbf{v}$ for some scalar σ, then σ is called the *eigenvalue* of $\mathbf{H}$ with corresponding (right) *eigenvector* $\mathbf{v}$. The *minor* of an $n \times n$ matrix $\mathbf{H}$ is the determinant of $k \times k$ matrix $\mathbf{M}$ obtained from $\mathbf{H}$ by deleting $n - k$ rows and $n - k$ columns of $\mathbf{H}$ where

$k \leq n$. The *cofactor* $\mathbf{C}_{ij}$ is defined as $\mathbf{C}_{ij} = (-1)^{i+j}\mathbf{M}_{ij}$ and is used to compute the *determinant of a matrix* as $|\mathbf{H}| = \sum_{i=1}^{n} h_{ij}\mathbf{C}_{ij}$.

The *conjugate* of $\mathbf{H} \in \mathbb{C}^{n\times n}$, denoted by $\mathbf{H}^*$, is obtained from $\mathbf{H}$ when each elements of $\mathbf{H}$ is replaced with its complex conjugate. The complex conjugate operator is distributive under complex addition and complex multiplication. The *conjugate transpose* matrix (also called the *adjoint* matrix) is defined as $\mathbf{H}^H = (\mathbf{H}^*)^T$ where $(\mathbf{AB})^H = \mathbf{B}^H\mathbf{A}^H$. A square matrix $\mathbf{H} \in \mathbb{C}^{n\times n}$ is a *hermitian* matrix if $\mathbf{H} = \mathbf{H}^H$. A square matrix $\mathbf{H} \in \mathbb{C}^{n\times n}$ is *unitary* if $\mathbf{H}^H\mathbf{H} = \mathbf{H}\mathbf{H}^H = \mathbf{I}$ or $\mathbf{H}^H = \mathbf{H}^{-1}$. Also, if $\mathbf{H}$ is unitary, then (a) $\mathbf{H}$ is *non-singular* (i.e., its determinant is nonzero and thus it has a matrix inverse) and $\mathbf{H}^H = \mathbf{H}^{-1}$, (b) $\mathbf{H}^H$ is unitary too, (c) the columns (the rows) of $\mathbf{H}$ form an orthonormal basis for $\mathbb{C}^n$. If the columns of $\mathbf{H}$ form an orthonormal set, then applying $\mathbf{H}$ to a vector does not change its length (i.e., $|\mathbf{Hs}| = |\mathbf{s}|$).

2.2 Noise Models at the Receiver

In digital wireless communication systems there are various sources of error-performance degradation such as noise, fading, and interference due to the filtering at the transmitter and receiver and also bandwidth-limited channel [12]. Noise is defined as unwanted and usually uncontrollable signal components that distort the intended signal. For example, noise signals can arise from harmonics of the natural frequency, atmospheric disturbances, and crosstalk from other communication systems.

Two common noises at the receiver are *thermal noise* and *quantization noise* [12]. Thermal noise in a circuit is primarily due to the random fluctuations of electrons. Quantization noise in a digital system is due to distortions caused by conforming to the finite word length that is available to represent a signal. It is commonly assumed that a noise source at the receiver emanates an equal amount of power per unit bandwidth at all frequencies. This implies that the noise, on the average, has just as much power per hertz in low-frequency fluctuations as in high-frequencies, which can go up to about 10^{12} Hz [12]. A corresponding random process $X(t)$ for representing noise has a flat PSD over all frequencies of interest (positive and negative) with fixed amplitude $G_X(f) = N_o/2$ W/Hz. The factor 1/2 has been included to indicate that half the power is associated with positive frequency and half with negative frequency. Noise that has such a uniform spectral density, is also called *white noise*, where "white" refers to the analogous case of white light which contains equal amounts of all frequencies within the visible band of electromagnetic radiation. Thermal

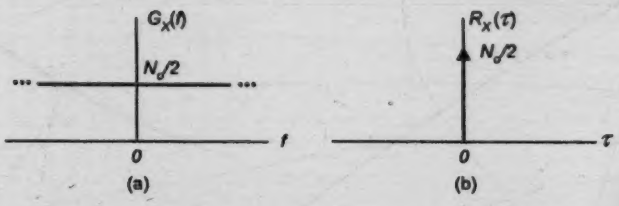

Figure 2.3: (a) PSD of white noise. (b) ACF of white noise.

noise and other sources of noise introduced by the amplifiers, mixers, and analog-to-digital converters are also ordinarily modeled as white noise [12]. Interestingly, quantization error has approximately a white power spectrum mainly due to the randomness of the signal [12]. Therefore, the total noise at the receiver is typically modeled as white noise.

It is generally assumed that there is no correlation between samples of the equivalent time-domain noise process $X(t)$, i.e., $\mathrm{R}_X(\tau) = 0$ for $\tau \neq 0$. For a real white noise process $X(t)$, the autocorrelation function is

$$\mathrm{R}_X(\tau) = \mathfrak{F}\{G_X(f)\} = \mathrm{E}[X(t)X(t+\tau)] = \frac{N_o}{2}\delta(\tau)$$

which is the delta function weighted by the factor $N_o/2$ and occurring at $\tau = 0$, as shown in Figure 2.3(b). As shown in Figure 2.3(a), white noise has a two-sided PSD of a constant amplitude $N_o/2$ over all frequencies $-\infty \leq f \leq \infty$. Hence the noise variance (that is, the average noise power, since the mean is zero) is

$$\sigma^2 = \mathrm{E}[X^2(t)] = \int_{-\infty}^{\infty} \frac{N_o}{2}\,\mathrm{d}f = \infty.$$

Although the variance for white noise is infinite, practical systems have finite bandwidth and hence the noise is typically band-limited to some $2W$ Hertz (i.e., $-W \leq f \leq W$). For example, the received signals (and noise) at the very front end of a receiver in communication systems involves processing by band-limited (typically narrow-band) filters. The PSD of a WSS white noise process whose frequency components are limited to $-W \leq f \leq W$ is shown in Figure 2.4(a). The average power in such an ideal case can be written as

$$\mathrm{E}[X^2(t)] = \int_{-W}^{W} \frac{N_o}{2}\mathrm{d}f = N_o W.$$

The ACF can be written as

$$\mathrm{R}_X(\tau) = \frac{N_o}{2}\int_{-W}^{W} e^{-j2\pi f\tau}\mathrm{d}f = \frac{N_o}{2}\frac{e^{-j2\pi W\tau} - e^{j2\pi W\tau}}{-j2\pi\tau} = \frac{N_o \sin(2\pi W\tau)}{2\pi\tau}.$$

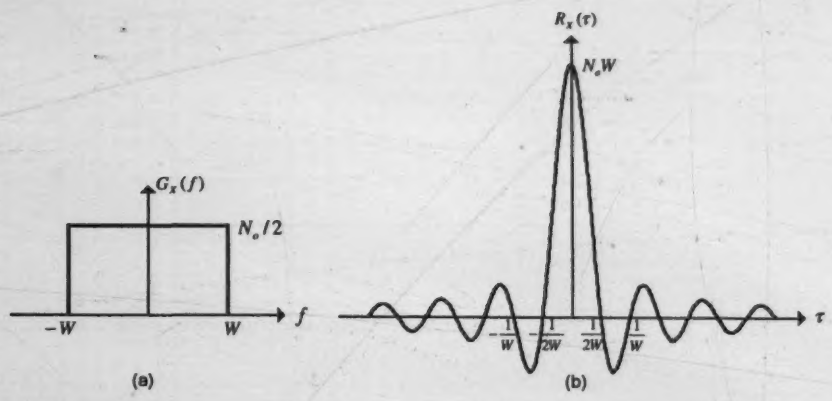

Figure 2.4: (a) PSD of bandlimited white noise. (b) ACF of bandlimited white noise.

As shown in Figure 2.4(b), $X(t)$ and $X(t+\tau)$ are uncorrelated at the zero-crossing $\tau = \pm k/(2W)$ where $k = 1, 2, \cdots$. This implies that samples in the time domain are uncorrelated (independent in the Gaussian case). If $F_s = 2W$, then the variance of the samples is $N_o F_s/2$.

White noise is commonly described as a Gaussian random process $X(t)$, and hence called white Gaussian noise (WGN), whose value (noise amplitude) at any arbitrary time t is statistically characterized by the Gaussian PDF

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma_X^2}} \exp\left(-\frac{(x-m_X)^2}{2\sigma_X^2}\right)$$

where σ_X^2 is the variance of X. The *normalized* or *standardized Gaussian density function* of a zero-mean process is obtained by setting the standard deviation $\sigma_X = 1$. The normalized PDF is shown in Figure 2.5. Figure 2.5 shows that most probable noise amplitudes are those with small (positive or negative) small values.

The main reason that the Gaussian distribution is often chosen as the system noise model is due to the *Central Limit Theorem* (CLT) [14]. The CLT can be explained as follows. Assume that $\{X_i\}$, $1 \le i \le N$, is a set of N i.i.d, zero mean random variables, with finite variance σ_X^2. For convenience we use the normalized random variable

$$Z_i = \frac{X_i - m_X}{\sigma_X}, \quad i = 1, 2, \cdots, N \tag{2.17}$$

with zero mean and unit variance. The normalized (by $1/\sqrt{N}$) random variable Y is defined as the sum

$$Y = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} Z_i \tag{2.18}$$

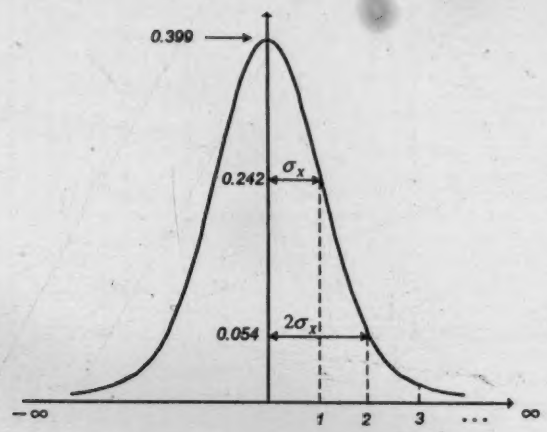

Figure 2.5: Normalized Gaussian PDF.

also has zero mean and unit variance. In the limit as $N \rightarrow \infty$, the CLT implies that the CDF of Y approaches the Gaussian distribution. Although it is assumed that random variables in the sum are identically distributed, the assumption can be relaxed by summing independent random variables each with PDF $f_{X_i}(x)$ and finite variance $\sigma^2_{X_i}$ [13].

The WGN noise model has been widely used in the modeling and verification of many communication systems [12]. For example, the *narrow-band noise* appearing at the output of digital filters at the receiver is commonly modeled as a Gaussian random process [15]. Narrow-band noise $X(t)$ is usually represented in terms of its in-phase and quadrature components, similar to narrow-band signals of the form $X(t) = X_i(t)\cos(2\pi f_c t) - X_q(t)\sin(2\pi f_c t)$. Since $X(t)$ is Gaussian, then the complex-valued Gaussian RP $X(t)$ consists of two jointly Gaussian real-valued processes. By *jointly Gaussian* we mean that any arbitrary set of samples of the real and imaginary parts is a Gaussian set of random variables [11].

In addition to modeling white noise at the receiver using a Gaussian process, *memoryless* channels are also modeled using a Gaussian process. Under a memoryless channel assumption, a scaled or attenuated amplitude version of the original signal is received at the receiver. The widely used communication channel model is the *additive white Gaussian noise* (AWGN) channel in which there is assumed to be a noise signal *superimposed* on the desired signal. Note that since samples of AWGN are independent, the noise effects each transmitted symbol independently.

One of the standard dimensionless parameters used in the error rate performance eval-

uation of digital communication systems is the *signal-to-noise ratio* (SNR), defined as the ratio of the average signal power to the average noise power. Being able to accurately estimate the receiver noise and SNR at the receiver improves the receiver's ability to make correct symbol decisions (probability of error), and hence increase the rate of reliable data transmission. A related figure of merit, E_b/N_o, is sometimes used where E_b is the *bit energy* (i.e., binary signal power S times bit time T_b) and N_o is the *one-sided noise density* (i.e., noise power N divided by the bandwidth W). Thus

$$\frac{E_b}{N_o} = \frac{S\,T_b}{N/W} = \frac{S/R}{N/W} = \frac{S}{N}.\frac{W}{R} \qquad \frac{\text{Joule}}{\text{Watt per Hz}} = \frac{\text{Watt.s}}{\text{Watt.s}}$$

Thus E_b/N_o is just signal-to-noise ratio normalized by the bandwidth and bit rate. The smaller the required E_b/N_o, the more efficient is the detection procedure for a given probability of error [12]. If P_T is the average transmitted power, the average energy per symbol can be written as $E_s = P_T T_s = q\,E_b$, where $q = \log_2 M$ and M is the size of the constellation. The received power P_R to noise PSD and the received bit-energy E_b to noise PSD are related as

$$\frac{P_R}{N_o} = \frac{E_b}{N_o} R = \frac{E_s}{N_o} R_s.$$

Thus

$$\frac{E_s}{N_o} = \log_2 M.\frac{E_b}{N_o}.$$

2.3 Wireless Channel Effects

In a typical urban area or indoor environment, the height of a transmitter antenna is often lower than many of the surrounding structures. Thus, a direct path or a *line of sight* (LOS) path between the transmitter and the receiver is often absent. Due to the processes of *reflection* (which occurs when a waveform meets an object that is much larger than the signal's wavelength), *diffraction* (which occurs when the surface encountered by the signal has irregularities such as sharp edges), and *scattering* (which occurs when the medium contains a large number of objects near the same size as the signal's wavelength) from objects in the path [16], multiple copies of the transmitted signal, called *multipath* signal components or rays, arrive at the receiver via several paths with different *angle of arrivals* (AOAs), time delays, and amplitude. More importantly, changes in the path length by Δd over a short time interval Δt causes a phase shift

$$\Delta\phi = \frac{2\pi\Delta d}{\lambda} = \frac{2\pi v \Delta t \cos\alpha}{\lambda}$$

where v is the velocity of the mobile unit (MU) and α is the AOA. Clearly, as the path length changes by a wavelength λ (30 cm at 1 GHz), the signal phase changes by 2π. Due to the significant phase changes, scattered components may add constructively at one location but add destructively at a location just a short distance away, according to their relative arrival times, amplitudes, and phases. The received signal envelope (level), not averaged over an area, fluctuates rapidly and randomly about the *local* mean over a short period of time or travel distance (typically over distances of about half a wavelength). When the signal power drops significantly, the channel is said to be in a fade and this phenomenon is called *small-scale fading* [17] as shown in Figure 2.6. The occasional deep amplitude fades coincide with rapid phase variations. For example, small-scale fading can attenuate the signal by 40 dB when the mobile moves as little as half a wavelength.

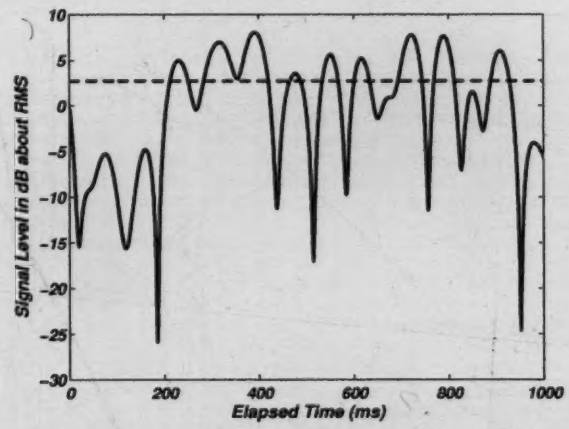

Figure 2.6: Typical simulated Rayleigh fading at the receiver.

If the transmission medium changes or if there is a relative motion of the antennas, the path length and/or geometry changes by Δd and each multipath signal component experiences an apparent shift in frequency, called a *Doppler shift*. The *Doppler frequency* is defined as

$$f_d = \frac{1}{2\pi}\frac{\Delta\phi}{\Delta t} = \frac{v\cos\alpha}{\lambda} = \frac{f_c v\cos\alpha}{c}$$

where f_c is the carrier frequency, $c \approx 3 \times 10^8$ m/s is the free-space velocity of the electromagnetic wave, and α is the direction of motion of the mobile with respect to the direction of multipath signal arrival. The motion of the MU will introduce changes in the channel at the rate of f_d Hz. For a constant mobile velocity, as f_c increases, the Doppler shift becomes

larger. If a sinusoidal signal at the carrier frequency f_c is transmitted, the received signal spectrum, called the *Doppler spectrum*, will have components lying in the range of $f_c - f_d$ to $f_c + f_d$. If the receiver is moving toward the transmitter, i.e., $-\pi/2 \leq \theta \leq \pi/2$, the Doppler shift is positive (i.e., the apparent received frequency $f = f_c + f_d$ is increased); otherwise, if the receiver's movement reverses direction then the Doppler shift is negative. Relative to the carrier frequency, the Doppler shift is typically quite small, but relative to baseband frequencies it can be relatively large.

Fluctuations in the received power are not the only effects of fading. Fading may also affect the shape of the pulse as it is being transmitted through the channel [18, 19]. If the received multipath components are resolvable [20], then multipath effects can result in the broadening of the transmitted pulse, leading to *inter-symbol interference* (ISI), where the pulses of adjacent symbols interfere at the symbol sampling times. It should be noted that the small-scale fading is caused by changes in phase rather than by path attenuation since the path lengths change by only a small amount over small distances. However, if the mobile moves over larger distances ($\gg \lambda$), due to the changes in terrain features, the received signal strength can attenuate significantly. Fading over a large distance, called *large-scale fading*, may be mitigated by the use of power control, for example, while small-scale fading will introduce the need of an equalizer that is capable of removing the time-varying ISI introduced by the multipath propagation.

To evaluate the performance of wireless communication systems in laboratories, a channel simulator must faithfully model both the large-scale and small-scale effects of time-varying propagation environments. Mainly, two approaches are utilized for modeling multipath fading channels: ray-theoretical modeling [21] and impulse-response modeling [22]. The *ray-theoretical model* illuminates essential characteristics of the channel based on geometric propagation theory and physical rays caused by reflections and diffractions. However, the high computation and lack of detailed terrain and building databases make these models difficult to use [23]. By far the most popular channel simulation models are *stochastic parametric models*. In this approach, the channel impulse response is characterized by a set of deterministic and random parameters. The values of the parameters and the probability distributions governing their behavior are selected according to empirical measurements. A multipath fading channel is commonly modeled as a linear time-varying (LTV) system and can be fully described by its impulse response [22, 24–26]. The complex impulse re-

sponse $c(t,\tau)$ is a low-pass equivalent model of the actual real band-pass impulse response

$$c(t,\tau) = \sum_{\ell=0}^{L(t)-1} a_\ell(t) e^{j\phi_\ell(t)} \delta\Big[t - \tau_\ell(t)\Big] \tag{2.19}$$

defined as the response observed at time t to an impulse applied at time $t-\tau$, where τ is the delay parameter, t is the time, and $L(t)$ is the number of *resolvable* multipath components [20]. The ℓ-th signal component experiences a different path environment which will determine the amplitude a_ℓ, carrier phase shift ϕ_ℓ, time delay τ_ℓ, AOA α_ℓ, and Doppler shift f_d. In general each of these parameters are time-varying. The amplitude $a_\ell(t)$ is usually modeled as a Rayleigh-distributed random variable as

$$f_X(x) = \frac{x}{\sigma_X^2} e^{-x^2/2\sigma^2}, \ \ x \geq 0$$

while the phase shift $\phi_\ell(t)$ is uniformly distributed. Note that the channel model in Equation (2.19) does not consider the AOA of each multipath component. It is usually assumed that the scatterers surrounding the mobile station are about the same height as or are higher than the mobile. This implies that the received signal at the mobile antenna arrives from all directions after bouncing from the surrounding scatterers. Under these conditions, the Gans assumption that the AOA is uniformly distributed over $[0, 2\pi]$ is valid [25]. The classical Rayleigh fading envelope with deep fades approximately $\lambda/2$ apart arises from this model [26].

The effects of multipath channel and noise on transmitted signals in a wireless communication system are shown in Figure 2.7. The transmitted baseband signal $s(t)$ will convolve with an L-ray multipath fading channel and then AWGN will be added to produce the output samples $z(t)$. The signal observed at the receiver can be written as

$$z(t) = s(t) * c(\tau, t) + n(t)$$

where $c(\tau,t)$ is the time-varying channel's impulse response to represent the impact of the channel on the transmitted signal $s(t)$, and $n(t)$ is AWGN. The equivalent lowpass received signal is given by

$$z(t) = \sum_{\ell=0}^{L-1} a_\ell(t) e^{-j\phi_\ell(t)} s(t - \ell T_c) + n(t)$$

where L resolvable paths are spaced at T_c time intervals. Wireless channel modeling will be discussed in detail in Chapter 4 and Chapter 5. Decoding, which is the process of mapping the received signal $z(t)$ into one of the possible transmitted symbols, will be further discussed in Chapter 6.

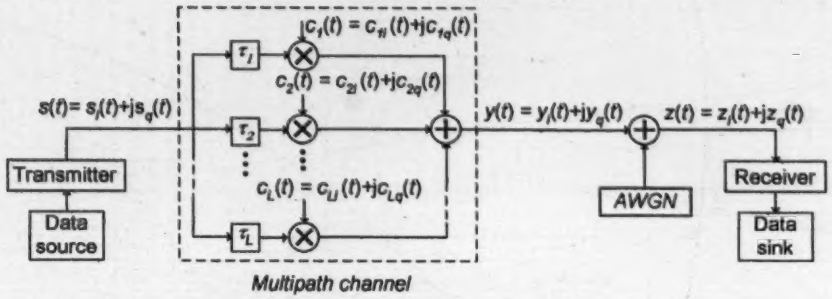

Figure 2.7: Block diagram of wireless communication systems.

2.4 High-Throughput Wireless Communication Systems

It is dangerous to put limits on wireless
(G. Marconi, 1932).

Consider a band-limited channel of bandwidth W Hz in which the signal is corrupted only by AWGN having a single-sided power spectral density of N_o watts/Hz. Claude Shannon defined the *channel capacity* to be the maximum error-free average data rate that a channel can support [27]. Shannon showed that for AWGN channels, the channel capacity is given by

$$C = W \log_2 \left(1 + \frac{S}{N}\right) = W \log_2 \left[1 + \frac{E_b}{N_o} \cdot \frac{R}{W}\right] \tag{2.20}$$

where C is the channel capacity in bps. Clearly the requirement for higher data rates directly translates into a wider bandwidth requirement and/or higher transmitted power which may exceed the design constraints. Since the radio spectrum is a limited (and regulated) resource, more efficient spectral utilization is attractive. *Spectral efficiency* is usually defined as the number of bits per second that can be transmitted per Hz of bandwidth. Let $E_b = S/C$ denote the energy per information bit at the receiver. Using Equation (2.20) the SNR E_b/N_o can be expressed as:

$$\frac{E_b}{N_o} = \frac{W}{C}(2^{C/W} - 1) \tag{2.21}$$

where C/W is the maximum achievable spectral efficiency. The simplest way to increase the spectral efficiency is by increasing E_b/N_o, which in turn implies increasing the transmitted signal power, or increasing the transmitter and receiver antenna gain, or decreasing the receiving system noise, or some combination of these measures [12]. It seems that data transmission at rates beyond the Shannon limit is only possible by increasing the capacity of the transmission channel itself.

In wireless communications, transmitted signals that are attenuated and undergo fading can have a severe impact on system performance. One effective approach to combat fading and increase the channel capacity is to deploy multiple antennas at the transmitter and/or receiver. Multiple antenna communications is popularly known as *space-time* (ST) *wireless* or wireless using *smart antennas*. Such multiple antenna systems can theoretically increase capacity by up to a factor equal to the minimum of the numbers of transmit and receive antennas [2]. The key idea in multiple antenna systems is that if several paths have channel coefficients that are *statistically independent*, it is unlikely that they will all fade together, so the probability of unreliable detection is greatly lessened. *Diversity* refers to the existence of two or more signal paths that fade independently. Various *spatial diversity* (or antenna diversity) schemes have been utilized in single-input multiple-output (SIMO) deployments [2]. Multiple-input single-output (MISO) systems have used *transmit diversity* schemes [2]. Utilizing diversity techniques in SIMO and MISO communication structures provides SNR gain, increases spectral efficiency on multipath channels, mitigates the effects of fading, and increases the channel capacity, especially when multiple antennas are also available at the receiver. However, the capacity increase may not be sufficient for future requirements of wireless communication systems.

MIMO communication systems with n_T transmit antennas and n_R receive antennas were introduced to provide diversity gain, array gain, interference reduction, and multiplexing gain [2]. The resulting diversity order increases by $n_T \times n_R$ in which the depth of fades reduces considerably and the mean signal level increases, i.e., the signal suffers less from deep fades. In addition, in richly-scattered multipath wireless channel, deploying multiple antennas at both the transmitter and receiver can achieve high data rates without increasing the total transmission power or bandwidth. The capacity of MIMO systems over richly scattered channel has been shown to grow linearly up to $\min(n_T, n_R)$ fold [2]. This maximum data rate increase is limited by the richness of the multipath environment (i.e., correlated fading across receive antennas). To assure independent fading, the receive and transmit antennas must be sufficiently separated in space and/or polarization to create independent propagation paths. The asymptotic behaviour of the ergodic capacity of multi-antenna systems is shown in Figure 2.8. MIMO is a key element in 802.11n, which is an emerging standard for next-generation of 802.11 that could boost throughput to 100 Mbit/s while maintaining backward compatibility with existing 802.11$a/b/g$ devices.

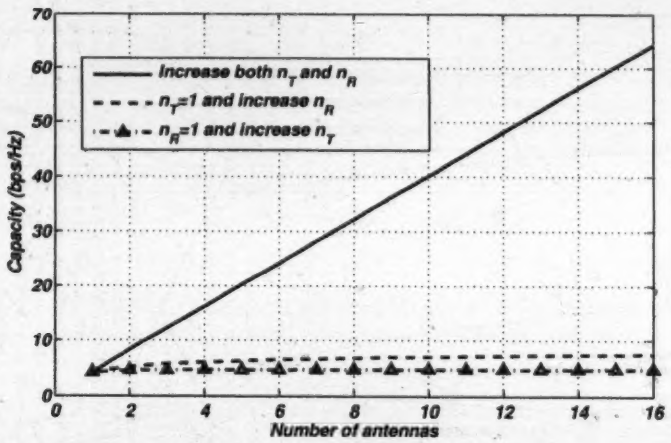

Figure 2.8: Ergodic capacities of uncorrelated multiple antenna systems for SNR=18 dB.

2.5 Algorithm Efficiency

In the evolutionary domain of wireless signal processing, careful analysis of proposed alternative algorithms is essential. For example, numerous models have been proposed for time-varying wireless channel models during the past three decades [28–39]. However, some of the well-known models do not generate correct statistical properties of free-space propagation [34, 37]. Unfortunately, they have been used in many channel simulators and thus likely have led to inaccurate performance evaluations.

Another important decision that a designer must make is that given several alternative algorithms for a given task, which one should be considered for implementation. Specifically, for signal decoding at the receiver, numerous algorithms with various bit error rate (BER) versus SNR performance characteristics have been proposed. In addition to reliability of the algorithm, which is usually measured in the average BER versus SNR, other characteristics of an algorithms such as numerical stability and computational complexity are important. For example, heuristic MIMO decoding algorithms have at least cubic computational complexity in the number of antennas while some other techniques have exponential complexity [2]. Hence, some of the published decoding algorithms may not be practical for real time implementation even for a moderate number of antennas and low-order digital modulation schemes.

One important challenge is, therefore, to minimize the computational complexity and

increase the efficiency of the decoding algorithm at the receiver. An important question that arises is how to assess the efficiency of a signal processing algorithm? The efficiency of a decoding algorithm relates to the amount of computation (i.e. the complexity) required to decode the received signals to achieve an acceptable BER. Most of the proposed decoding approaches for MIMO systems consider only the average number of elementary operations, such as complex additions and multiplications per decoded symbol or bit, as a measure of the complexity of the algorithm [40]. The execution time of an algorithm on a sequential machine is usefully expressed as [41]

$$execution\ \ time = IC \times CPI \times CT \tag{2.22}$$

where IC, CPI and CT represent the instruction count, the average number of clock cycles per instruction and the clock period, respectively. Equation (2.22) shows that the execution time depends directly on the clock period (technology-dependent), the instruction count (depends on the *instruction set architecture* (ISA), operating system, programming style, programming language and compiler, etc.), and the average number of clock cycles per instruction (depends on the implementation of the ISA). Consequently, the performance of the algorithm can be expressed as $IPC \times$ *clock frequency* / IC, where IPC is the average number of instructions per clock cycle ($IPC = 1/CPI$). Reducing the instruction count in the implementation tends to increase the rate at which the algorithm can be completed. However, it is not sufficient to only consider the number of basic operations when comparing the efficiency of algorithms: there are other important factors. A more useful efficiency measure should also consider such characteristics as the degree of randomness or determinism in the execution behaviour, the maximum amount of limited resources (such as memory) required to execute the algorithm, the numerical robustness against quantization and round-off errors, the degree of concurrency, and the energy required by the hardware. The randomness of an algorithm can be defined as the dependence of the algorithm's execution behaviour on the input data set. For instance, sphere decoding (SD) is a data-dependent symbol decoding algorithm where, depending on the channel conditions and noise variance, one can obtain very different run times [42].

The degree of concurrency generally depends on the method that the overall computation is broken down into finer subtasks and divided among computational units for parallel execution. If an algorithm is parallelizable, then the efficiency of it can be defined using

Amdahl's law [43] as

$$E = \frac{T_1}{n_p T_p}$$

where T_1 is the sequential execution time of the algorithm, T_p is the time spent by a parallel algorithm that uses n_p PEs [44], and the ratio T_1/T_p is called the *speed-up*. This law states that the efficiency of the parallel machine (algorithm) is measured by the overall utilization of the n_p functional units or fraction of time the n_p processing units are busy. In other words, the performance gain is limited by the fraction of time that algorithm uses n_p PEs. More efficient algorithms achieve higher utilization of the processor's resources (such as memories and functional units) at any given point of time.

Typically, physical layer signal processing algorithms of wireless communication networks contain abundant data parallelism as identical operations are performed repeatedly on incoming streams of input data. The data and instruction level parallelism can be exploited to increase the IPC rate. Thus, another efficiency measure is the degree to which an algorithm can effectively exploit the available parallelism. If the target microarchitecture is known *a priori*, designers/compiler may exploit parallelism from an existing algorithm to increase the computation speed. For example, most high-performance *digital signal processors* (DSPs) provide parallelism in their ISA [41]. Typically they exploit instruction-level parallelism and subword parallelism within a processing unit and data-level parallelism across a limited number of processing modules. If the target platform is an *application-specific integrated circuit* (ASIC), in contrast to fixed-ISA architectures, the number of processing units can be scaled to meet real-time constraints of the algorithm. Also, designers may modify the algorithm in such a way that can be efficiently mapped on a simple parallel architecture. For example, if a parallel algorithm requires the transmission of a data among PEs for every few numerical operations it performs, then the communication time is likely to dominate its execution time. Ideally, the structure of the algorithm should match the communication structure between the PEs of a parallel architecture. If the target platform is ASIC or FPGAs, then it is reasonable to modify the algorithm to reduce hardware complexity at the expense of an acceptable performance penalty.

2.6 Base-Band Signal Processing Platforms

The demand for multimedia mobile communications with better QoS has been steadily increasing. Correspondingly, wireless standards are evolving to support more users with

higher data rate communications than before. Some communication systems, such as MIMO, rely on computationally-intensive signal processing algorithms at the receiver to support high data rate transmissions. The demanding high performance requirements of these systems have forced manufacturers to modify their signal processor architectures to meet the necessary processing throughput. DSP architectures have been evolving towards higher clock frequencies, more instruction-level parallelism (ILP), and the inclusion of domain-specific functional units. For example, the Texas Instruments TI C6x processors [45] use a very long instruction word (VLIW) architecture while the TI C5x [46] includes a dedicated hardware unit to accelerate the add-compare-select (i.e., butterfly) operation required by the Viterbi decoder in global system for mobile communications (GSM) cellular networks. Similarly, FPGAs have evolved to enable computationally intensive base-band signal processing.

The dynamic environment of wireless communications also favors programmable, or in some way reconfigurable, solutions. For example, it is highly desirable that the processing platform be flexible enough to execute different algorithms. A link adaptation algorithm, which usually resides in the media-access controller layer, provides the switching between the diversity and spatial multiplexing modes of operation, depending on the channel conditions. Another example is a base station that can seamlessly switch between different wireless standards (e.g., GSM and IS-95). Of course, each of those standards requires different physical layer algorithms where algorithmic parameters, such as the coding rate and modulation order, need to be configured based on the propagation environment. Another advantage of programmable solutions is that they can be updated in the field with only software changes [47]. Moreover, flexible architectures facilitate the tracking of evolving standards, protocols, and services with the same basic hardware components. Decoupling the evolutionary flow of upgrading wireless standards for better services from hardware design also extends the time-in-the-market.

Considering the strict performance requirements and the advantages of flexibility, communication system designers have a vast range of semiconductor technologies to choose from when developing such a system. Three main processing platforms have been commonly used: (a) software-programmable VLSI circuits such as DSPs and micro-controllers; (b) hardware-programmable circuits such as FPGAs, and (c) ASICs. High-performance DSPs have become an integral component in base-stations to perform part of the required processing. Even though they provide flexibility through software programmability, how-

ever, their use in portable terminals is not widespread due to their relatively high power consumption. In high data rate wireless applications, where the required processing exceeds the processing power of DSPs, the ASIC is an efficient solution to off-load part of the processing. A dedicated co-processor can be utilized to accelerate the execution of time-critical kernels. ASICs are more power-efficient and support the often tremendous amount of processing needed to implement baseband signal processing. Even though ASICs can provide some level of flexibility, they still need to be co-ordinated by a micro-controller. This leads to the notion of embedded DSP and micro-controller cores integrated along with the ASIC engine to provide a better solution.

DSPs and ASICs are by far the most popular processing modules in wireless communications systems. FPGAs are reconfigurable due to their programmable fabric that trades additional silicon area for its flexibility "in the field". They succeed in bridging the performance and flexibility gaps between DSPs and the custom design approach by providing post-fabrication programmability. Fixed-point arithmetic computational units with parameterized precisions, variable-length registers, and numerous dedicated signal processing cores on-chip provide a flexible target platform for the efficient realization of digital signal processing algorithms that were formerly implemented on digital signal processors. FPGAs also increasingly provide distributed arithmetic, parallel processing, and high data bandwidth between processing fabric and on-chip memory blocks, making them ideal for the realization of computationally-intensive physical layer algorithms. The more recent FPGAs include programmable processors to execute complex algorithms and control functions. For example, embedded Nios soft processors are available in Altera's Stratix devices [48] and Power PC processors are available in Xilinx Virtex II Pro devices [49].

Each processing platform has its advantages, and the optimum implementation strategy will vary depending upon application requirements and the cost. DSPs are well suited to arithmetic-intensive tasks, with conditional processing. For example, a DSP can simply execute a standard floating-point C program that has various if-then-else clauses such as the protocol stacks of communications systems. The complexity will affect the program length and hence the execution time. The performance of a DSP is mainly restricted by the clock rate and the number of independent operations that can be performed by the limited number of functional units. In contrast, FPGA designers are not constrained by fixed datapaths or the relatively small number of processing elements that are available in DSPs. They can add variable length registers, several specialized execution units to handle

more threads simultaneously, and wide data buses to reach the required data bandwidth for specific algorithms. When mapping an algorithm with high degree of concurrency onto the array of processing elements on FPGAs, the number of computing resources and the flexibility in their possible connection allow the designer to provide an optimized mix of task-level parallelism and instruction-level parallelism for the particular algorithm.

Similar to DSP program libraries, FPGAs include various hard IP cores and usually come with a set of predefined soft IP cores that can be synthesized and implemented efficiently on FPGAs. Thus, FPGAs offer the flexibility to implement exactly what is required for a given application using highly parameterized building blocks. For example, Xilinx FPGAs can provide an opportunity for greater computational capacity than programmable DSPs by executing above 600 billion multiply-accumulate (MAC) per second compared to the 4.8 billion MAC/sec on the Analog Devices TigerSHARC DSP [50]. Also, several digital clock manager cores are embedded in FPGAs that provide clock frequency synthesis (flexible clock multiplication and division). This core allows adjusting the sample rate at various nodes in the system which may not be possible in DSPs. Thus, FPGAs can be utilized as co-processors to implement compute-intensive multi-rate signal processing algorithms.

We can conclude that the wireless infrastructure poses a set of stringent implementation requirements that are often contradictory. The evolutionary and adaptive nature of wireless standards requires flexible solutions, which are usually most easily solved using software-programmable architectures. Yet these solutions do not offer the required performance and energy-efficiency for high data rate communication systems. The latter can be offered by custom/semi-custom designs, but these tend to lack the required flexibility. Hence, dedicated ASICs can be utilized to perform some of the highly repetitive tasks required by such systems.

Even though reusing a common hardware across different functions will reduce the overall costs, running various tasks of communication systems on a single hardware platform can easily result in an inefficient implementation. Due to the different structure of algorithms, some algorithms may be poorly matched to a particular hardware architecture. For example, communication protocol stacks are complex control-dominated software code, more suitable to be implemented on micro-controllers or DSPs. Also, depending on the life time of a processing block, if it is short it is more suitable to be implemented on DSPs, since to be executed by FPGAs, it must be implemented in the core of the FPGA,

which takes resources.

As a result, the implementation platform for wireless systems must balance the contradictory requirements of high-performance, flexibility, low energy consumption, and low cost [51]. Hence, the architectural platform for wireless applications will typically include heterogenous collections of computational modules, each of which targets a particular function. Given the wide range in the architectural options, it is important to elaborate trade-offs in the flexibility-efficiency-performance-cost space when selecting the components for this heterogenous fabric. It seems that system-on-a-chip (SOC) technology, which combines a wide range of processors such as DSPs, ASICs, and configurable logic on a single die, is an efficient processing platform for the next generation of wireless systems.

2.7 Rapid Prototyping in Digital Design

In a highly innovative market, wireless systems have very short production cycles. In addition, evolving standards, such as the third and fourth generations, continuously add new features and modes to improve one of the fundamental goals such as maximizing the transmission rate and minimizing the error rate. Achieving these goals requires continuous redesigns and simulations of baseband algorithms and changing design parameters during the design flow. Since the time to market is directly influenced by the time of design and verification, the design productivity can be increased by speeding up the design cycle.

Verification of alternative signal processing algorithms typically starts in their floating-point representations. At this stage algorithms assume few architectural or implementation details and are not realizable in their original double-precision form. In addition, to limit the required hardware resource requirements (and hence limit the cost and power consumption), most signal processing algorithms are implemented with fixed word-length precision. Relying entirely on floating-point simulation results during algorithm development can lead to failure in hardware implementation, as in the case of high performance radio local area network (HiperLAN) standard [52]. Hence, a transformation from the floating-point to a fixed-point system is required. Using finite-precision simulation, sufficiently large dynamic range (to avoid overflow) and sufficiently wide precision (to bound the accumulated round-off errors) of each operand can be obtained. Also, the impact of finite-word-length arithmetic and quantization on the performance and the complexity of computation must be verified against that of the original floating-point implementations. However, the bit-true simulation of low error rate communication systems would be significantly slow. To imple-

Table 2.1: Parameters of different Xilinx FPGAs.

Device Family	Slices	Multiplier Blocks	XtremeDSP Slices	Block Memories	User I/O Pads	PowerPC Blocks
Xilinx Virtex2P XC2VP100-6	44,096	444	*NA*	444	1,164	2
Xilinx Virtex4 XC4VFX140-11	63,168	*NA*	192	552	896	2
Xilinx Virtex4 XC4VSX55-11	24,576	*NA*	512	320	640	*NA*

ment a fixed-point simulator, one may write/invoke C++ libraries for fixed point operands (objects) and may overload required operators. Typically, the fixed-point constructs cannot be efficiently mapped to the general-purpose architecture of the host machine [53]. Moreover, the complexity of signal processing algorithms and large number of samples required for a faithful verification slows down software-based simulations.

A hardware-based *rapid prototyping platform* (RPP) can not only speed up the bit-true simulation, it can also integrate algorithm development and implementation as early as possible [54]. Results from hardware simulation can be immediately fed back to the design process thereby avoiding the significant delay often experienced in the traditional design verification process. A RPP can be used to cope with rapidly evolving and increasingly complex communication standards, to assess alternative candidate algorithm and implementation strategies, to evaluate the design complexity and feasibility, to identify the real-time bottlenecks of the proposed algorithms, and to determine the required hardware resources.

In this thesis we used FPGAs from two well-known producers of programmable logic devices: Xilinx Inc. and Altera Corp. They build families of programmable devices ranging from low-cost FPGAs (such as Cyclone and Spartan), to high and medium density FPGAs (such as Stratix II and Virtex II Pro), and embedded processors (such as the Nios and PowerPC processors). While they sometimes have different terminologies for different features (for example *logical elements* in Altera devices versus *slices* in Xilinx FPGAs), the devices from both vendors have many common features such as local and global interconnect, hierarchical clocking, register chains, memory blocks, and various dedicated processing units.

Some of the major features of Xilinx FPGAs are summarized in Table 2.1. Each programmable *slice* contains two function generators, two storage elements, arithmetic logic gates, large multiplexers, and fast carry look-ahead chain. A *configurable logic block* (CLB) is made up of four slices. While Virtex family parts have dedicated multipliers, Virtex 4 parts contain XtremeDSP slices that each include one 18×18-bit 2's complement signed multiplier, adder logic, and a 48-bit accumulator, operating at up to 500 MHz. Each mul-

Table 2.2: Parameters of Altera Stratix EP1S80F1508C6 FPGA.

Logic Elements	M-RAM Blocks	M4K Blocks	M512 Blocks	DSP Blocks	User I/O Pins	Embedded Multipliers
79,040	9	364	767	22	1,238	176

tiplier and accumulator can be used independently. These blocks are designed to support efficient and high-speed DSP applications. Each BlockRAM is an 18-Kbit true dual-port RAM, programmable from 16K $\times$ 1 to 512 $\times$ 36, in various depth and word width configurations. Each port is totally synchronous and independent. The BlockRAMs are cascadable to implement large embedded memories. Additionally, a built-in first-in, first-out (FIFO) memory is supported in the Virtex-4 FPGAs. Each PowerPC 405 core is a 32-bit Harvard-architecture microprocessor with a five-stage execution pipeline, a 16-KB Level 1 instruction cache, a 16 KB Level 1 data cache, and operates at up to 450 MHz.

Altera Stratix devices provide three different RAM block sizes to implement true dual-port memory and FIFO buffers: 512 Kbits M-RAM blocks, 4K bits M4K Blocks, and 512-bit M512 blocks. High-speed DSP blocks provide dedicated implementation of fast multipliers (faster than 300 MHz), and MAC functions. Nios is a 32-bit pipelined and configurable embedded reduced instruction set computer (RISC) processor. It operates at up to 125 MHz, with 16-bit instructions, and 32-bit or 16-bit configurable data path. Important specifications of Altera Stratix EP1S80F1508C6 device are given in Table 2.2.

While each FPGA has resources that can be configured in device-specific ways, our designs were implemented using synthesizable HDL and independent of these features. This allows us to target other hardware platforms and ASICs. However, to obtain best resource utilization and performance, we have also implemented our designs using specific features of FPGAs. For example, a four-input lookup table in Xilinx Virtex FPGAs were configured as a 16-bit shift register for compact implementations of pseudo-random number generators. These registers were also used to implement delay lines with various lengths for multipath fading channel simulators. LUTs were also configured as distributed memories and act to provide compact implementations of registers, instead of using limited number of flip-flops in each slice. Even though implementing pipelined parallel multipliers using configurable slices requires a relatively large number of slices, dedicated 18 $\times$ 18-bit multipliers are extremely useful for implementing MAC operations. Moreover, 48-bit DSP blocks are useful when implementing PEs of DSP-RAM parallel processor where each PE performs a 16 $\times$ 16 multiplication and a 48-bit accumulation operation. Various configu-

rations of dual-port block memories allow compact and efficient storage of constant values such as sine and cosine values along with the coefficients of polynomials and digital filters. Block memories are also efficient resources for implementing delay lines with longer delays compared to implementations using LUTs configured in SRL mode. Furthermore, an array of block memories and dedicated DSP blocks are useful for implementing PIM processor architectures. The remaining chapters will discuss device-specific features in more detail.

Chapter 3

Gaussian Variate Generators

One of the many applications of random variates with a Gaussian PDF is to model noisy natural phenomena. For example, a sequence of GVs is commonly used to model additive noise or variations in signal attenuation in the propagation channel. Also, multipath fading channel simulators use independent Gaussian samples to generate Rayleigh fading variates (i.e., signal attenuation coefficients). Another important application of GVs is to evaluate the performance of communication systems in the presence of additive noise at the receiver. The error rate performance of the system will depend on the channel model and noise values introduced by different sources such as the non-linearity of the filters and quantization errors. While the distribution of distortion values may be difficult to characterize analytically, it could be verified by MC simulation. In such a case, the overall effect of distortion from all functional blocks can be modeled together and there is no need to model them separately.

While a communication system can be characterized through the symbol error rate (SER) versus SNR relationship, for high SNR regions (corresponding to very low BERs) such a characterization requires very long-running MC simulations. For example, consider a digital communication system that is designed to achieve a BER of no more than 10^{-14}. This means that on average, 10^{14} symbols must be processed for each erroneous symbol in a MC simulation of the system. One usually requires at least 100 such error events if the BER is to be estimated with reasonable statistical significance. In addition, approximately 10 samples per symbol interval are typically required to successfully represent waveforms in the simulation. Thus roughly 10^{17} samples must be processed. The generation of 10^9 GVs, using an optimized software simulator written in C, takes about 2.5 hours on a dual Pentium processor running at 3.0 GHz with a 1-MB L2 cache. By extrapolation, the gener-

ation of 10^{17} GVs at this rate would take about 27,000 years. Note that some recent standards give maximum allowable BERs of 10^{-12} for specified SNRs (e.g., IEEE 802.11ae 10 Gbit/sec Ethernet). Thus maximizing the achievable GV generation rate is crucial for validating the BER performance of upcoming communication systems, and software-based GV generation is no longer adequate.

The low-BER characterization problem imposes additional challenges on PDF accuracy. Evaluating a communication system at a BER of 10^{-15} implies that approximately one in every 10^{15} received bits should be errored. The $Q(\cdot)$ error function, that is commonly used for determining the probability of error of uncoded systems, for 10^{-15} [14] is a value that approaches 8.0. This means that random variates near the center of the distribution do not contribute significantly to the probability of error since their small values are readily tolerated by any system with that low a BER. Rather it is the GVs with a value of 8σ or larger, where σ is the standard deviation of the Gaussian distribution, that will be the dominant source of errors. Therefore, for a MC simulation, the PDF of generated random numbers must be accurately close to the Gaussian PDF at the high σ regions (the tails of the PDF). Since the tail of Gaussian PDF decays exponentially, generating accurately-distributed GVs with large σ values is quite challenging.

Hardware-based GVGs using analog devices [55–57] and digital components [58–63] have shown significant speedups compared to software implementations. However, digital implementations tend to be more desirable than analog implementations due to their predictable and controllable behavior, and because they can reproduce exactly the same pseudorandom sequence of variates in successive runs.

In this chapter we present different approaches to efficiently implementing GVGs based on the Box-Muller algorithm [64]. When implementing a compact and accurate GVG on FPGAs, the following objectives need to be considered:

- Tail accuracy: The normal distribution is an open-ended distribution in which extreme values occur with increasingly small probabilities. A GVG for low BER characterization must be able to generate accurately distributed GVs, especially at the high σ values.

- Statistical correctness: The generated variates must pass standard tests for statistical properties, such as independence and accurate Gaussian distribution. The quality of the proposed design must be supported by standard goodness-of-fit statistical tests.

- Hardware efficiency: Ideally, a hardware realization should minimize the number of required FPGA resources while achieving an acceptably fast variate generation rate. Compactness is a more important factor when the available hardware resources are especially limited, such as in a relatively low-density FPGA.

- Flexibility: The design should be parameterized and easy to modify to satisfy different design constraints, such as maximum resource utilization and minimum sampling rate.

The rest of this chapter is organized as follows. Section 3.1 reviews Gaussian distribution properties, describes several algorithms for generating GVs, and also compares related work on digital GVG implementations. Section 3.2 evaluates different pseudo-random number generators (PNGs) and compares their statistical properties. Section 3.3 discusses different techniques for the implementation of trigonometric functions. Sections 3.4 describes trade-offs involved in implementations of the Box-Muller algorithm. Specifically, three different realizations for implementing the BM algorithm are proposed and implementation results are discussed. A compact and accurate hardware GVG that is described occupies only 1% of the Xilinx Virtex-II XC2V4000-6 FPGA and operates at 253 MHz, generating 506 million Gaussian variates per second within a range of $\pm 9.41\sigma$. The design can be easily configured to achieve higher tail accuracy at a small cost in extra hardware but with slightly decreased operating rate. Various standard statistical tests are applied to this GVG to verify its statistical characteristics. Implementation results verify that the PDF of the generated variate samples accurately matches the Gaussian PDF, even at the tails of the distribution. Also, the generated variates pass numerous standard goodness-of-fit statistical tests. Section 3.5 is dedicated to the analysis of goodness-of-fit tests and simulation results. Concluding remarks appear in Section 3.6.

3.1 Gaussian Distribution, Algorithms, and Related Work

A standard Gaussian random variable $Z \in \mathbb{R}$, with a mean of zero and a variance of one, has the probability density function $f_Z(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$ [14]. More generally, a Gaussian random variable $X = \sigma_X Z + m_X$ is completely characterized by its mean m_X and variance σ_X^2 and this property is denoted as $X \sim \mathcal{N}(m_X, \sigma_X^2)$. The PDF of X can be expressed as

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma_X^2}} \exp\left(-\frac{(x - m_X)^2}{2\sigma_X^2}\right)$$

where $1/\sqrt{2\pi\sigma_X^2}$ is the normalization constant that is chosen so that the area under $f_X(x)$ is unity. The CDF is defined as the probability that a variable X has a value less than or equal to x, and it is expressed in terms of the PDF as

$$F_X(x) = \frac{1}{\sqrt{2\pi\sigma_X^2}} \int_{-\infty}^{x} \exp\big(-\frac{(y - m_X)^2}{2\sigma_X^2}\big)\, \mathrm{d}y$$

for which there is no closed-form expression. Note that $\mathrm{d}F_X(x)/\mathrm{d}x = f_X(x) \geq 0$ and $F_X(x)$ grows monotonically from 0 to 1 such that the $F_X(x)$ values are uniformly distributed. Two important properties of the Gaussian variables are: (1) If $X \sim \mathcal{N}(m_X, \sigma_X^2)$ and if a and b are real numbers, then $aX + b \sim \mathcal{N}(am_X + b, (a\sigma_X)^2)$; and (2) if $X \sim \mathcal{N}(m_X, \sigma_X^2)$ and $Y \sim \mathcal{N}(m_Y, \sigma_Y^2)$ are independent normal random variables, then their sum is also normally distributed with $W = X + Y \sim \mathcal{N}(m_X + m_Y, \sigma_X^2 + \sigma_Y^2)$.

When evaluating the performance of communication systems, it is often necessary to determine the area under the tail of the Gaussian PDF, the so-called *tail probability* [12]. The probability under the tail of the standard Gaussian PDF ($m_X = 0$ and $\sigma_X^2 = 1$) can be written as

$$Q(x) = \Pr(X > x) = 1 - F_X(x) \approx \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-y^2/2}\, \mathrm{d}y$$

where $Q(x)$ is the integral of the tail of the Gaussian PDF and is shown in Figure 3.1. Since $Q(x)$ cannot be evaluated in closed form (i.e., cannot be summed and expressed as a mathematical formula), a tabular format is typically used. An approximation for $x > 3$ can be written as [12]

$$Q(x) \approx \frac{1}{x\sqrt{2\pi}} e^{-x^2/2}.$$

The $Q(x)$ is related to the error function $\mathrm{erf}(x)$ and the complementary error function $\mathrm{erfc}(x)$ is related by

$$Q(x) = \frac{1}{2}\mathrm{erfc}\big(\frac{x}{\sqrt{2}}\big) = \frac{1}{2}\Big[1 - \mathrm{erf}\big(\frac{x}{\sqrt{2}}\big)\Big]$$

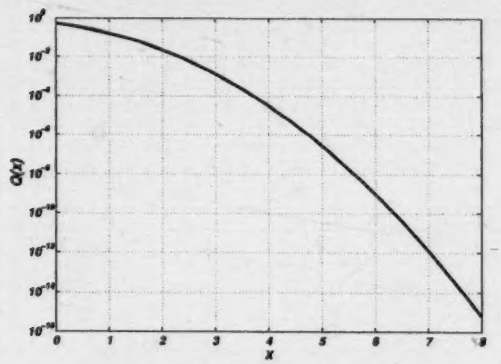

Figure 3.1: Plot of $Q(x)$.

where

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-y^2} \mathrm{d}y$$

and

$$\text{erfc}(x) = 1 - \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-y^2} \mathrm{d}y.$$

This probability is commonly used for determining the probability of error in the presence of Gaussian noise. For example, when detecting orthogonal signals at the receiver, the probability of error is reported to be [12]

$$P_E = Q\left(\sqrt{\frac{E_b}{N_o}}\right) \tag{3.1}$$

which shows that the P_E decreases with increasing energy per bit E_b and/or decreasing noise power.

The fundamental importance of the normal distribution is due to the CLT. According to the CLT, the sum of a relatively large number of i.i.d random variables with a finite variance will be asymptotically normally distributed [14]. If u is uniformly distributed over the interval $[a, b]$ with $m_U = (a + b)/2$ and $\sigma_U^2 = (b - a)^2/12$, then one can show that if $u_i \in [0, 1]$ then

$$X = \sqrt{\frac{12}{k}} \left(\sum_{i=1}^{k} u_i - \frac{k}{2} \right), \tag{3.2}$$

where $m_X = k/2$ and $\sigma_X = \sqrt{k/12}$, is approximately Gaussian distributed with $\mathcal{N}(0, 1)$. The drawback of this method is that $X \in [-\sqrt{3k}, \sqrt{3k}]$ instead of being unbounded within $(-\infty, \infty)$. Thus X is distributed as $\mathcal{N}(0, 1)$ only asymptotically as k goes to infinity. The

CLT-based technique may be acceptable in many applications because 99.7% of the observations fall within $\pm 3\sigma_X$ of the mean. However, accurately generating Gaussian samples beyond three standard deviations from the mean requires summing up a much larger number of samples implying much more hardware or computation time. Therefore, a purely CLT-based approach is impractical for the accurate implementation of GVGs.

The standard approach for generating nonuniform random variates such as Gaussian random numbers is to produce uniform random numbers first and then to transform them to obtain the desired PDF [65]. For example, the inversion method [66] transforms uniform random variables $U \in [0, 1]$ into a Gaussian variable X by approximating the nondecreasing inverse of the Gaussian CDF as $X = F_X^{-1}(U)$. Even though $F_X(x)$ cannot be inverted analytically, the inverse CDF, also called the quantile function, can be expressed in terms of the inverse error function $\sqrt{2}\ \mathrm{erf}^{-1}(2u - 1)$. The GVG in [67] uses a look-up table to store the CDF inverse. However, it is reported in [68] that this approach cannot produce Gaussian variates with accurate high-σ values.

Other popular approaches are the rejection-acceptance schemes such as the polar method and the Ziggurat algorithm [69, 70]. The polar algorithm generates two independent GVs at a time. It involves finding a random point within the unit circle by generating uniformly distributed points within the $[-1, 1] \times [-1, 1]$ square and then rejecting any points outside of the circle. The polar algorithm is given in Algorithm 1 where "*rand*()" generates a pseudo-random number (PN) between $[0, 1]$, and x_1 and x_2 are two independent normally distributed elements. The Ziggurat method is a special case of the rejection method. The

Algorithm 1 The polar algorithm.

1: **while** $n \geq 1$ **do**
2: $u_1 = rand(); \ u_2 = rand();$
3: $v_1 = 2u_1 - 1; \ v_2 = 2u_2 - 1;$
4: $n = v_1^2 + v_2^2;$
5: **end while**
6: $x_1 = v_1\sqrt{-2\ln n/n}; \ x_2 = v_2\sqrt{-2\ln n/n};$

main advantage of this algorithm comes from the fact that for a high percentage of the generated numbers, no relatively slow $\ln(\cdot)$ operation is necessary. The rejection-based approaches have been generally used in software implementations (for example, the polar algorithm is used in older versions of Matlab - before MATLAB 5; - the Ziggurat algorithm is used in Matlab 5; and a modified Ziggurat algorithm is used in MATLAB 6), however, due to their conditional *if-then-else* assignment instructions and also the required $\ln(\cdot)$ compu-

tation, the Ziggurat algorithm has received less attention as the basis for efficient hardware implementation [71].

Another method was proposed by Box and Muller and is now well-known as the Box-Muller (BM) [64] algorithm. This algorithm transforms pairs of uniformly distributed numbers into samples from a two-dimensional bivariate normal distribution. The inputs to the BM algorithm are two independent uniformly-distributed PNs, $u_1, u_2 . \in [0, 1]$. The outputs are two independent samples, x_1 and x_2, from $\mathcal{N}(0, 1)$. The transformation involves multiplying $f(u_1) = \sqrt{-2\ln(u_1)}$ by $g_1(u_2) = \sin(2\pi u_2)$ and $g_2(u_2) = \cos(2\pi u_2)$ to yield x_1 and x_2, respectively. This can be justified by solving for u_1 and u_2 where

$$u_1 = \exp\left(\frac{-(x_1^2 + x_2^2)}{2}\right) \text{ and } u_2 = \frac{1}{2\pi}\arctan(\frac{x_2}{x_1}).$$

The joint PDF of X_1 and X_2 can be written as [64]

$$f_{X_1,X_2}(x_1, x_2) = \frac{1}{2\pi}\exp\left(\frac{-(x_1 + x_2)}{2}\right) = \frac{1}{\sqrt{2\pi}}e^{\frac{-x_1^2}{2}} \cdot \frac{1}{\sqrt{2\pi}}e^{\frac{-x_2^2}{2}} = f_{X_1}(x_1)f_{X_2}(x_2) \tag{3.3}$$

which shows that X_1 and X_2 are independent variates from $\mathcal{N}(0, 1)$.

The normality of the resulting distributions in the BM algorithm depends on the statistical properties of the PNG and the accuracy of the $f(\cdot)$ and $g(\cdot)$ computations. The maximum representable value of the generalized GVs depends on the precision of the uniform variables u_1 and u_2. Thus the effects of the finite precision representation of variables on the accuracy of fixed-point arithmetic must be considered carefully. Assume that $u_1 \in (0, 1)$ is represented in the unsigned fixed-point format, which we will denote by $\mathcal{Q} < WL >$, where WL is the word length. Because of the similarity of the sine and cosine functions, $g_1(\cdot)$ and $g_2(\cdot)$ can share the same hardware, thus we will use $g(\cdot)$ to denote the implemented function, either $g_1(\cdot)$ or $g_2(\cdot)$. Since $|g(\cdot)| \leq 1$, the maximum absolute value of a GV beyond 1.0 is determined by the function $f(\cdot)$. Thus, in practice u_2 is less critical to the final accuracy and so $g(\cdot)$ can use a lower bitwidth. The closer that the value of u_1 is to 0, the greater is the value of $f(\cdot)$ and, therefore, the greater is the magnitude of the generated GV. One way to ensure that $|x| \leq \varsigma\sigma$, for some desired value ς, is to choose the precision WL of u_1 to be large enough so that $\sqrt{2WL\ln(2)} \geq \varsigma\sigma$. One can readily verify that for 32-bit precision in u_1, a variable range $|x| \leq 6.66\sigma$ can be obtained. Table 3.1 presents the maximum representable value of $f(u_1)$, namely $\sqrt{2WL\ln(2)}$, for various precisions of u_1. The precision of the PNG can always be adjusted to obtain a Gaussian PDF with any desired accuracy in the tails.

Table 3.1: Maximum absolute value of a GV for various precisions of u_1.

WL	8	16	24	32	48	64
Maximum $\lvert f(u_1)\rvert$	3.33	4.70	5.76	6.66	8.15	9.41

Table 3.2: Published FPGA Implementations of a GVG.

Design	[61][a]	[75]	[62]	[77]	[79]
Maximum deviation	4.0σ	4.8σ	6.0σ	5.0σ	6.0σ
Output rate (MGVs/sec)	24.5	245	133	165	155
Clock freq. (MHz)	98	245	133	165	155
Number of slices	437	480	2514	702	770
Resource utilization	8%	2%	10%	3%	3%
On-chip memory blocks	0.5	5	2	5	6

[a]Reference [61] used the Altera Flex10KE FPGA; the four other designs used a Xilinx Virtex-II XC2V4000-6 FPGA.

Digital implementations of high-quality PNGs and accurate realizations of the logarithm, square root and trigonometric functions have been investigated extensively over the last three decades [72]. On the other hand, the BM algorithm is a reliable technique to generate GVs with large values (accurately in the tail of PDF). Consequently, the BM transformation has been used in many FPGA [61,62,73–77] and parallel processor realizations [78]. Table 3.2 summarizes the major characteristics of various published GVG implementations.

3.2 A Closer Look at PNGs

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin (Von Neumann, 1951).

The Monte Carlo (MC) technique is the basis for simulating systems that are driven by at least one input signal that is modeled as a stochastic process [13] - as opposed to *deterministic* algorithms. Other inputs can be modeled using analytical techniques or based on empirical measurements. Consequently, the key to a MC simulation, is the generation of sequences of random numbers which represent the sampled values of the input signal (i.e., a random process). The accuracy of MC simulation results depends on different parameters such as the size and statistical distribution of input samples and also system modeling assumptions and approximations.

PNGs have been used in MC simulators as either random sample generators or utilized to be transformed to other random sequences with a different (e.g., Gaussian) distribution.

Consequently, MC simulations rely on the quality of PNGs (how closely they resemble truly random sequences). It is unfortunate that PNGs with actually rather poor statistical properties are frequently recommended in many texts and are used in many applications. Even many widely used PNGs that perform well in standard statistical tests for randomness are known to be inadequate for MC simulations [80,81].

A PNG uses a deterministic algorithm that has a state that evolves in a finite state space $\mathcal{S}$ according to a recurrence of the form $s_i = f(s_{i-1}), i \geq 1$. The initial state $s_0 \in \mathcal{S}$ is called the *seed*, and $f : \mathcal{S} \rightarrow \mathcal{S}$ is the *transition function*. At step $i \geq 0$, the generator outputs $u_i = g(s_i)$, where $g : \mathcal{S} \rightarrow [0,1]$ is the output function. The null hypothesis for a PNG can be explained as follows: the $\{u_i\}$ are i.i.d random variables that are uniformly distributed over the $[0,1]$ interval if for each i and d, the vector $\mathbf{u_i} = \{u_i, u_{i+1}, \cdots, u_{i+d-1}\}$ is uniformly distributed over the d-dimensional unit hypercube $[0,1)^d$. The independence property implies that subsequences of the generated PNs $u_0, u_1, \cdots$ should be statistically independent. For a correlation test between consecutive PNs $u_0, u_1, \cdots$ in the interval $[0,1)$, either overlapping d-tuples $\mathbf{u_i} = \{u_i, u_{i+1}, \cdots, u_{i+d-1}\}$ or non-overlapping d-tuples $\mathbf{u_i} = \{u_{id}, u_{id+1}, \cdots, u_{id+d-1}\}$ can be constructed, and then the distribution of finite sequences $\mathbf{u_i}$ for each i in d-dimensional unit hypercube $[0,1)^d$ for small values of d (e.g., $d \leq 6$) can be assessed. The task is to measure how well $\mathbf{u_i}$ is uniformly distributed. For example, the generator should produce non-overlapping pairs $(u_{2i}, u_{2i+1}), i = 0, 1, \cdots$ (i.e., members of the even subsequence $u_0, u_2, \cdots$ should be independent of odd neighbors $u_1, u_3, \cdots$). Undesirable correlations between consecutive random numbers will lead to deviations of the empirical distribution function of $\mathbf{u_i}$ from the uniform distribution, in some dimension d. Regularities in generators and subtle correlations between PNs can compromise MC results [82].

An ideal random number generator would provide numbers that are uncorrelated (of central importance for many stochastic simulations), would satisfy relevant statistical tests of randomness, would have a large repetition period (this period limits the number of samples that we can use safely), would generate a deterministic sequence that could be changed by adjusting an initial "seed" value, and would rapidly generate numbers using minimal hardware resources.

There are two main types of random number generators for producing sequences of PNs: *linear* PNGs and *nonlinear* PNGs [65]. A common example of linear PNGs are the *linear congruential generators* (LCGs) [83]. LCGs are widely used in many applications

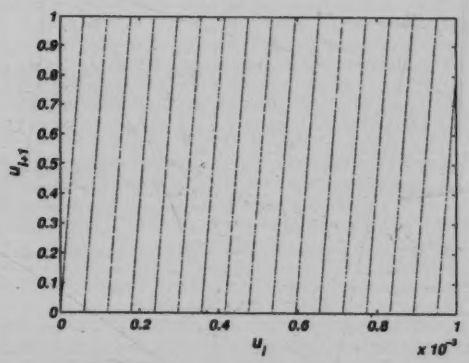

Figure 3.2: The two-dimensional distribution of 10^7 PNs pairs (u_i, u_{i+1}) generated with MLCG recursion with $x_0 = 314519$, $a = 16807$, and $m = 2^{31} - 1$, when a small portion of u_i-axis is magnified.

such as ANSI-C RAND (32-bit precision) function. They are based on the integer recursion

$$x_i = (a\, x_{i-1} + b) \mod m$$

where the multiplier a, increment b, and modulus m are constants. If a is a primitive root modulo m and m is prime, then the period of the generator is $\rho = m - 1$ [84]. These generators can be further classified into *mixed* ($b > 0$) and *multiplicative* ($b = 0$) types, usually denoted by LCG(a, b, m) and MLCG(a, m). An LCG generates a sequence of pseudo random integers $x_1, x_2, \cdots$ between 0 and $m - 1$; for MLCG the lower bound is 1. Each x_i can be scaled down into the $u_i \in [0, 1)$ interval. For a fast and convenient implementation of LCGs on computers, a modulus that is a power of 2 is commonly used. An intrinsic property of LCGs is that the distribution of d-tuples $\mathbf{u}_i = \{u_i, u_{i+1}, \cdots, u_{i+d-1}\}$ over all possible points in $[0, 1)^d$ lies on a relatively small number of parallel hyperplanes [65]. The intuitive reason is that there are 2^{dWL} points in the unit d-cube with coordinates that are exactly representable as WL-bit binary fractions, where WL denotes the word length in bits. These points lie on 2^{WL} hyperplanes at a separation of 2^{-WL}.

Figure 3.2 plots the two-dimensional distribution of 10^7 PNs pairs (u_i, u_{i+1}) generated using MLCG recursion with seed $x_0 = 314519$ when a small portion of u_i-axis is magnified. A lattice structure is clearly visible in the smallest case $d = 2$. In addition to regular lattice structures, the other known defect of this generator is that it produces correlated low-order bits as well as long-range correlations for intervals that are a power of 2. To avoid the problem of nonrandomness in the low order bits, the modulus m sometimes is chosen to be

a prime number. For example, IBM computers use a MLCG with $a = 7^5$ and $m = 2^{31} - 1$ where the period $2^{31} - 1 \approx 2.15 \times 10^9$ is relatively short [85].

To increase the rather short period of LCGs, the order of the linear recursion can be increased as

$$x_i = (a_1 x_{i-1} + \cdots + a_k x_{i-k}) \mod m,$$

that is called the *multiplicative recursive congruential generators* (MRGs) [83] with order $k > 1$, $a_i \in \mathbb{Z}_m = \{0, 1, \cdots, \}$ and $u_i = x_i/m$. For prime m and appropriately chosen a_i's, the sequence has a (maximal) period length $\rho = m^k - 1$. This can be achieved with only two non-zero a_i coefficients, i.e., $x_i = (a_r x_{i-r} + a_k x_{i-k}) \mod m$, which makes implementation faster. The special case $k = 2$, $a_1 = a_2 = 1$ leads to the Fibonacci generator $x_i = (x_{i-2} + x_{i-1}) \mod m$, whose statistical properties are rather poor. The Lagged Fibonacci generators (LFGs), denoted by LFG$(r, k, m, \odot)$ [83], are initialized with r integers $x_1, \cdots, x_r$ and use the recursion

$$x_i = (x_{i-r} \odot x_{i-k}) \mod m$$

where indexes r and k denote the lags, $r > k$, and $\odot$ denotes one of the binary operations $+, -, \times$ or the exclusive-or operation $\oplus$. For the common cases of addition or subtraction modulo 2^{WL}, the maximal period with suitable choices of r and k is $\rho \approx (2^r - 1)2^{WL-1} \approx 2^{r+WL-1}$. The standard Unix generator RANDOM and also the PNG in the Numerical Recipes implementation [86], which uses LFG$(55, 24, 10^9, -)$, are two examples of this kind of PNG. However, it is shown in [81] that LFGs using the operations of $(+, -, \oplus)$ can give poor performance unless the lag is very large (of order hundreds). One could choose multiplicative LFGs that scramble the regularity and increase the apparent randomness even for small lags. However, the resource utilization can be relatively large and the sampling rate is limited by the efficiency of the multiplier.

Linear feedback shift register (LFSR) generators, also called *Tausworthe generators* (TGs), are another important class of PNGs that are based on the theory of primitive trinomials of the form $p(x) = x^r + x^k + 1$ [83]. The sequence LFSR$(r, k, \oplus)$ is defined by

$$x_i = x_{i-r} \oplus x_{i-k}$$

The maximal possible period of $\rho = 2^r - 1$ is achieved when the primitive polynomial $p(x)$ divides $x^\rho - 1$ but for no smaller value of ρ. This condition can be met by choosing r to be a Mersenne prime, that is a prime number r for which $2^r - 1$ is also a prime. XOR-based

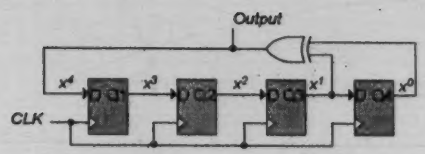

Figure 3.3: Fibonacci implementation of a 4-bit LFSR.

LFSRs have enjoyed success in many hardware applications because of their compactness, but have been criticized as being among the worst PNGs and that they produce sequences that are inadequate as pseudorandom sequences in MC simulations [87]. The PNGs with power of two moduli are also worse than those with prime moduli. Lindholm [88] also showed that the recurrences based on polynomials with too few nonzero coefficients will have inferior statistics.

Nevertheless, LFSRs have been widely used in hardware implementations because they tend to be faster than generators that use multiplication and because the period increases exponentially with the width of the register. Consider the 4-bit LFSR defined by the characteristic polynomial $p(x) = x^4 + x + 1$, as shown in Figure 3.3. This LFSR generates a single bit every clock cycle as an output. If one obtains a multiple-bit PN from the outputs of the flipflops, the PNs would be highly correlated. When an $m > 1$-bit PN is required, one way is to accumulate a m-bit PN using a LFSR which is rather slow [76]. Another approach is to use several LFSRs operating in parallel with uncorrelated initial seeds. This may lead to poor utilization of FPGA resources since LFSRs require little combinational circuitry and thus the slices would not be fully utilized. The more recent FPGAs (e.g., Xilinx Virtex family) provide special configurations (such as the shift register look-up tables [89]) to configure LUTs as a shift register and hence use the resources more efficiently when implementing LFSRs. This feature is used in GVGs in [62, 76, 77]. However, to maximize design portability, one would rather not use such device-specific optimizations. A more efficient approach is to use a *multiple-bit leap-forward LFSR* (LF-LFSR) [90] as used in GVGs in [61,77]. This method is based on the observation that the LFSR is a linear system and a LF-LFSR can leap k steps in one clock cycle. The LFSR state transition can be written as

$$Q_1^{s+1} = Q_3^s \oplus Q_4^s, \quad Q_2^{s+1} = Q_1^s, \quad Q_3^{s+1} = Q_2^s, \quad Q_4^{s+1} = Q_3^s$$

Figure 3.4: Rule 90 for cellular automata.

which in vector format can be expressed as:

$$\begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix}^{s+1} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix}^{s} = \mathbf{T} \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix}^{s}$$

where $\mathbf{T}$ is the transition matrix. It can be readily shown that the state of the LFSR after k steps (clock cycles) can be expressed as $\mathbf{Q}^{s+k} = \mathbf{T}^k \mathbf{Q}^s$. The new circuit thus leaps k steps in one clock cycle. For example, a four-bit LF-LFSR can be obtained using the following equations:

$$\begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix}^{4} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix}$$

$$Q_1^4 = Q_1 \oplus Q_3 \oplus Q_4, \quad Q_2^4 = Q_1 \oplus Q_2, \quad Q_3^4 = Q_2 \oplus Q_3, \quad Q_4^4 = Q_3 \oplus Q_4$$

Cellular automata generators (CAGs) [91] evolve in discrete steps, where the next value of registers determined by its previous value and that of the neighbors. Depending on the extent of neighborhood, different register-based state machines are constructed using so-called the *cellular automata rules*. For example, the rules $s_i^+ = s_{i-1} \oplus s_{i+1}$ and $s_i^+ = s_{i-1} \oplus s_i \oplus s_{i+1}$ define the so-called "Rule 90" (as shown in Figure 3.4) and "Rule 150" cell types, respectively, where s_i denotes the current value of i-th cell. CAGs rely on the bit-level computation and interconnection and are of particular importance for fine-grained parallel processing and is commonly used in built-in self-test applications. It is shown in [92] that the same linear dependencies that exist in the output sequence of LFSRs are still encountered in sequences generated by CAGs. In many applications, such as stochastic modeling and MC simulations, the quality of statistical randomness is far more important than long period and maximum generation rate [65]. In MC applications *it is better to be slow than sorry.*

Marsaglia and L'Ecuyer [93, 94] proposed CTGs [95] which have randomness properties that has strong theoretical and empirical support. The CTG is constructed by taking an exclusive-OR of several TGs which yield sequences that have much less regular structure

than the corresponding sequences of their individual component generators. If the component generators are chosen properly, then the period of the combined generator will be the product of the periods of the components [65]. The uniformity and independence of the output sequence is typically assessed by equidistribution measures [96]. The pseudo-code of a 32-bit CTG with three components and $\rho \approx 2^{88}$ is given in the following [97]. A restriction applies when assigning initial seeds to $z_j, j = 1, 2, 3$ where $z_1 \geq 2$, $z_2 \geq 8$, and $z_3 \geq 16$. It is recommended that the initial seeds, z_j, should be chosen to be large independent values [98].

```
unsigned long z1, z2, z3, b;
double CTG88 () {
b = (((z1 << 13) ⊕ z1) >> 19);  z1 = (((z1 & 4294967294) << 12) ⊕ b);
b = (((z2 << 2) ⊕ z2) >> 25);   z2 = (((z2 & 4294967288) << 4) ⊕ b);
b = (((z3 << 3) ⊕ z3) >> 11);   z3 = (((z3 & 4294967280) << 17) ⊕ b);
return((z1 ⊕ z2 ⊕ z3) × 2.3283064365e − 10);
```

A 32-bit CTG with four components and $\rho \approx 2^{113}$ can be designed as given in the CTG113 pseudo code [97].

```
unsigned long z1, z2, z3, z4, b;
double CTG113 () {
b = (((z1 << 6) ⊕ z1) >> 13);   z1 = (((z1 & 4294967294) << 18) ⊕ b);
b = (((z2 << 2) ⊕ z2) >> 27);   z2 = (((z2 & 4294967288) << 2) ⊕ b);
b = (((z3 << 13) ⊕ z3) >> 21);  z3 = (((z3 & 4294967280) << 7) ⊕ b);
b = (((z4 << 3) ⊕ z4) >> 12);   z4 = (((z4 & 4294967168) << 13) ⊕ b);
return((z1 ⊕ z2 ⊕ z3 ⊕ z4) × 2.3283064365387e − 10);
```

Unfortunately, the CTGs still have a lattice correlation structure and will fail tests that are sensitive to linear interdependencies [85]. Coddington proposed in [81] to improve the statistics of linear PNGs by using a very long period (e.g. $\rho \approx 2^{200}$ or more). This recommendation is based on empirical experiences and no theoretical analysis has been provided yet. A 64-bit CTG with five components and $\rho \approx 2^{258}$ is given in the following where $z_j, j = 1, \cdots, 5$ are 64-bit variables. Table 3.3 compares the performance of three CTGs.

```
unsigned long long z1, z2, z3, z4, z5;
double CTG258 () {
unsigned long long b;
b = (((z1 << 1) ⊕ z1) >> 53);   z1 = (((z1 & 18446744073709551614) << 10) ⊕ b);
b = (((z2 << 24) ⊕ z2) >> 50);  z2 = (((z2 & 18446744073709551104) << 5) ⊕ b);
b = (((z3 << 3) ⊕ z3) >> 23);   z3 = (((z3 & 18446744073709547520) << 29) ⊕ b);
b = (((z4 << 5) ⊕ z4) >> 24);   z4 = (((z4 & 18446744073709420544) << 23) ⊕ b);
b = (((z5 << 3) ⊕ z5) >> 33);   z5 = (((z5 & 18446744073701163008) << 8) ⊕ b);
return ((z1 ⊕ z2 ⊕ z3 ⊕ z4 ⊕ z5) × 5.4210108624275221e − 20);
```

Table 3.3: Performance of three CTGs implemented on the Xilinx Virtex-II XC2V4000-6.

Output bitwidth	*Apprx. period*	*Slices*	*Clock frequency (MHz)*
32	2^{88}	72	425
32	2^{133}	124	422
64	2^{258}	205	419

Table 3.4: PNGs used in published GVG designs.

Design	*Type*	*Period*
[61]	LFSR	2^{60}
[75]	LF-LFSR	2^{190}
[62]	LFSR	2^{60}
[77]	LFSR	2^{52}
[79]	CTG	2^{88}

In contrast to linear PNGs, *nonlinear generators* with a prime modulus overcome the regular structure of d-tuples of consecutive numbers [99]. The importance of inversive PNGs stems from the fact that their intrinsic structure and correlation behaviour are strongly different from linear generators. Also, there is no sample size restriction as for linear generators [96]. There are several variants of nonlinear generators such as inversive congruential generators (ICGs), explicit-inverse congruential generators (EICGs), and combinations of them. The congruence $x_{i+1} = a\bar{x}_i + b \mod m$ with $i \geq 0$ and seed x_0 defines an ICG that generates a sequence of PNs in $\{0, \cdots, m-1\}$. A prominent feature of the ICG with prime modulus is the absence of any lattice structure. The EICG follows the recurrence $x_i = \overline{a(i + x_0) + b} \mod m$ and defines a sequence of PNs in $\{0, \cdots, m-1\}$. Compared to linear PNGs of the same size, nonlinear PNGs tend to be slower but their structure and correlation behaviour are quite different from that of linear PNGs. An efficient way of obtaining a nonlinear PNG with a long period is to combine several small nonlinear PNGs with relatively prime moduli [96]. Fast implementations of sufficiently small nonlinear generators can be obtained simply by precomputing and storing their output sequences. Other nonlinear generators have been proposed in the field of cryptography [100]. For example, the advanced encryption standard (AES) is a cryptographic algorithm that has been used as an encryption function of communication systems such as in optical links. AES includes a nonlinear PNG that could be used for stochastic simulation [100]. The implementation results in [101] show that although AES in is indeed a good candidate for PNG, the resource need can be unacceptable for many applications.

Table 3.4 gives the PNGs used in different recently published GVGs. One important

point worth mentioning is that the period of a generator limits the usable sample size. For linear PNGs, the square root of the period length has been reported to be a prudent upper bound for the usable sample size [65]. Consequently, the maximum usable sample size of the generators given in Table 3.4 is actually too small for MC simulations of digital communication systems that operate at a very low error rates. Nonetheless, the test results in [81, 96] verify that for comparable periods, the mixed combined generator remained useful for longer sequences than the linear PNGs. Also, among numerous tested PNGs [81], the LCG with at least 48-bit precision, the LFGs using multiplication, the CTG with long period, and non-linear PNGs seemed to be among the best PNGs that provide better randomness properties and the extremely large periods necessary for MC simulations. So they are prime candidates for PNG realizations. To reduce the regularity, we can additively combine a large linear CTG with the period length of $\rho \approx 2^{258}$ [94] that passes all known statistical tests and various MC tests [81] with a very small non-linear PNG.

3.3 Implementation of Trigonometric Functions

There are various standard approaches for approximating trigonometric functions. The choice of a method and a particular implementation depends on such requirements as throughput, latency, and area as well as the required accuracy. The accuracy is determined by the error of the approximation and by the roundoff errors that occur during the evaluation of the approximation.

A polynomial approximation based on the *Taylor series* expression is a well-known technique that can be used for infinitely differentiable functions. If f^i denotes the i-th derivative of $f(\cdot)$, then the Taylor series of $f(x)$ about x_0 can be expressed as

$$f(x) = f(x_0) + \sum_{i=0}^{\infty} \frac{f^i(x_0)}{i!}(x - x_0)^i.$$

The special case for $x_0 = 0$ is called the *Maclaurin series*. The $\sin(\cdot)$ and $\cos(\cdot)$ functions can be approximated as

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \quad \text{and} \quad \cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} \qquad \text{for all } x,$$

respectively [102]. Clearly, the greater the number of partial sums that are included in the series, the more accurate the approximation. Factoring terms can lead to a simpler hardware

implementation, as shown in the following approximation:

$$\begin{aligned}\sin(x) &\approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \\ &\approx x(1-(x^2)(1-(x^2)(1-(x^2)/(6\times 7))/(4\times 5))/(2\times 3)).\end{aligned}$$

Thus, we only need to calculate x^2 at any one step, as opposed to x^7, and we do not have to calculate all of the factorial terms. In general, implementation of trigonometric (and other functions such as $\ln(\cdot)$) using only a few terms of the Taylor expansion provides unacceptable accuracies [102].

Storing and later retrieving quantized values of the trigonometric function in an on-chip memory, so-called *table lookup*, is relatively fast, but the function accuracy is limited by the size of the on-chip memory. In fact the size of memory grows exponentially with the size of the input word, which confines this solution to relatively small input precisions (say ≤ 12 bits arguments). The table size can be reduced by exploiting the symmetry properties of the $\sin(\cdot)$ and $\cos(\cdot)$ functions. Instead of storing the quantized values of $g_1(u_2)$ and $g_2(u_2)$ over the full period $[0,1)$, the domain of $g_1(u_2)$ within only the initial quarter period $[0, 0.25]$ can be segmented uniformly into 2^{q_2} segments and the corresponding function values can be stored in a $2^{q_2} \times WL_2$-bit BlockRAM. Algorithm 2 shows the calculation of the $\sin(2\pi u_2)$ value using the quarter-size look-up table. Algorithm 3 calls the $\sin(2\pi u_2)$ algorithm to calculate $\cos(2\pi u_2)$.

Algorithm 2 Computation of $\sin(2\pi u_2)$, $u_2 \in [0,1)$

```
if u2 > 0.5 then
  sign = 1; u2 = u2 − 0.5;
else
  sign = 0;
end if
if u2 > 0.25 then
  u2 = 0.5 − u2;
end if
index = ⌊4u2(2^q2 − 1.0)⌋;
return {sign, BlockRAM[index]};
```

Another well-known technique for reducing the table size, while maintaining or increasing accuracy, is to use an *interpolation* technique. In this method, a readily evaluated polynomial of degree v, $p(x) = \sum_{j=0}^{v} a_j x^j$, is obtained by making its value coincide with the function at $v+1$ points (breakpoints). To obtain the coefficients a_j, the set of $v+1$ linear equations $y_i = \sum_{j=0}^{v} a_j x_i^j$, $0 \leq i \leq v$ can be solved, where x_i and y_i are the $v+1$ breakpoints. An alternative to fitting a polynomial of degree v through

Algorithm 3 Computation of $\cos(2\pi u_2)$, $u_2 \in [0, 1)$

```
if u2 > 0.75 then
  u2 = u2 − 0.75;
else
  u2 = u2 + 0.25;
end if
return sin(2πu2);
```

$v + 1$ breakpoints is to have different polynomials (of lower degree) through subsets of the breakpoints. This approach is called *piecewise* approximation. If the breakpoints are sufficiently close, then one may choose linear interpolation. If the values of $\sin(2\pi x)$ for two successive known points, x_0 and x_1, are $f(x_0)$ and $f(x_1)$, respectively, then a linear interpolation can be used to approximate the value x between breakpoints x_0 and x_1 as $\frac{f(x)-f(x_0)}{f(x_1)-f(x_0)} = \frac{x-x_0}{x_1-x_0}$. Since x is known, the linear interpolation polynomial $p(x)$ can be defined as $p(x) = f(x_0) + (x - x_0)\frac{f(x_1)-f(x_0)}{x_1-x_0}$, where $\frac{f(x_1)-f(x_0)}{x_1-x_0}$ and $f(x_0)$ are typically stored in a table for every segment. The error of the linear interpolation is defined as $\xi = \|f(\cdot) - p(\cdot)\|$. Figure 3.5 plots the squared error of the direct lookup table technique along with the squared error of the linear interpolation method. The mean square error (MSE) of direct table lookup with 1024 segments for the full cycle is about 10^{-5}; for the first order approximation approach with 64 segments for the quarter cycle, the MSE is about 10^{-9}. The above techniques provide different trade-offs between the computation speed, memory size, and computation accuracy. One commercial implementation, the Synopsys trigonometric IP core, approximates the sine function by segmenting one quarter period into 64 segments, and then uses linear interpolation and symmetry to approximate the cosine value over the full domain $(0, 2\pi)$ [103].

Another technique is the *multipartite table method* in which only table lookup and adders are utilized [104] for a piecewise linear approximation of the functions with up to 20-bit inputs. The idea behind the bipartite approach is to group 2^α input intervals into 2^β larger intervals, where $\beta < \alpha$, such that the slope of the segment is considered constant in every larger interval.

Another well-publicized technique for approximating trigonometric functions is the CORDIC algorithm [72, 105]. CORDIC stands for *COrdinate Rotation DIgital Computer* [106] is a bit-serial set of algorithms that was further expanded to compute other elementary functions such as logarithmic, exponential and square-root. The CORDIC computation algorithms use additive normalization (i.e., each iteration uses a table lookup, bit shifts, and

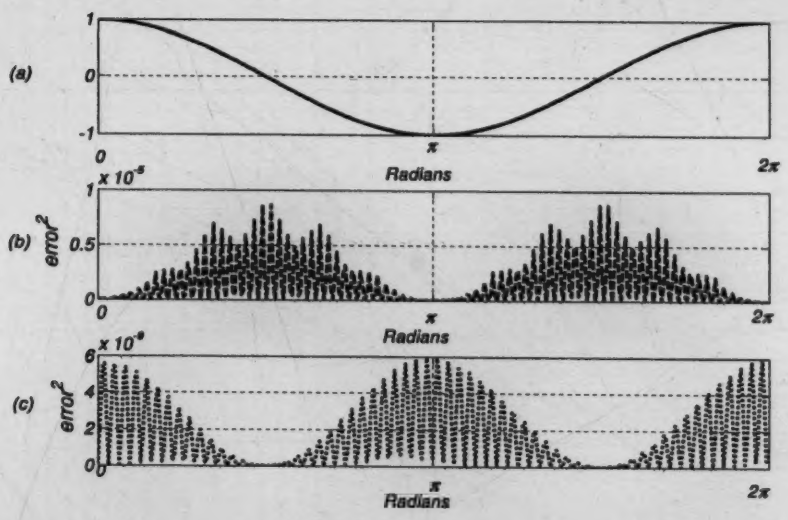

Figure 3.5: The cosine approximation error: (a) Ideal cosine function; (b) Squared error for the table look-up approximation; (c) Squared error for the first order approximation.

an addition operation to converge) which results in one bit convergence per iteration and hence a linear complexity $\mathcal{O}(n)$. It is particularly well-suited for low-cost applications. Xilinx provides a CORDIC IP core for implementing trigonometric equations [107]. A fully parallel 16-bit fixed-point configuration with single-cycle data throughput can be run at 200 MHz and requires 646 slices on a Xilinx Virtex2P XC2VP100-6 FPGA. Xilinx's word-serial architecture, configured with the maximum degree of pipelining, uses only 285 slices, but operates at the significantly lower frequency of 136 MHz. Redundant or high-radix CORDIC algorithms require more hardware but reduce the required number of iterations. Trade-offs thus exist between the complexity of each iteration and the number of iterations.

3.4 Gaussian Variate Generator Implementation

Accurate and efficient implementation of the $f(u_1)$ in the BM function is more challenging than realizing periodic sine and cosine functions. In this section, we propose three approaches for realizing $f(u_1)$ and hence producing three alternative GVGs. The first approach uses iterative convergent algorithms to calculate the $\ln(\cdot)$ and square-root operations. The second approach is based on non-uniform quantization and table lookup schemes. The third proposed method utilizes polynomial curve fitting, an efficient hierarchical domain

segmentation, and a novel scaling scheme to maintain the accuracy of the computation. The third technique provides an optimized datapath with respect to the noise generation rate and overall resource utilization on the FPGA, while achieving accurately Gaussian statistics right into the distribution tails.

3.4.1 Implementation of a GVG Using Iterative Algorithms

As discussed in Section 3.3, one solution for calculating the logarithm and square root operations is based on series expansion. It is known that this approach may result in unacceptable PDF accuracy if an insufficient number of leading terms of the Taylor series is utilized [58]. Another well-known approach for calculating differentiable functions, such as division, reciprocal, and square root, are the digit recurrence methods. These techniques produce linear convergence where each iteration produces one new digit of the result. As a consequence, the number of iterations for a desired accuracy depends linearly on the precision of u_1.

To compute $f(u_1)$, first $\ln(u_1)$ is calculated using the multiplicative algorithm for reciprocal [72] and then the square root of $-2\ln(u_1)$ is approximated with a *digit recurrence algorithm* [105]. Let $u_1 \in [0.5, 1)$ be a PN that has the fixed-point representation denoted by $Q\langle WL_1, WF_1\rangle$, where $WL_1 = 32$ and $WF_1 = 31$. Since $1/u_1$ can be approximated as $1/u_1 \approx \Pi_{j=0}^{WL_1}(1 + s_j 2^{-j})$, where $s_j \in [-1, 0, 1]$ [72], then $\ln(u_1)$ can be written as $\ln(u_1) \approx -\sum_{j=0}^{WL_1} L_j$ where $L_j = \ln(1 + s_j 2^{-j})$ [72]. Algorithm 4 shows an iterative procedure for calculating $y = \ln(u_1)$. The values for s_j are obtained from the multiplicative normalization and the values of L_j can be obtained from a table. The recurrence $y[j+1] = y[j] - L_j$ updates the value of y in every iteration j, where $0 \leq j \leq WL_1$, and the final result $y[WL_1] \approx \ln(u_1)$ is ready after $WL_1 + 1$ iterations. The absolute error of the $\ln(u_1)$ approximation, produced by this convergent algorithm, is bounded by 2^{-WL_1} [72]. However, in a finite-precision representation, the truncation (or rounding) error must also be considered. As shown in Algorithm 4, If u_1 is less than 0.5, u_1 should be scaled up and, consequently, the result of the algorithm would need to be rescaled. Note that if u_1 is equal to 0, the corresponding numerical value is set to the smallest representable number, 2^{-31}. Scaling up can be performed by a simple shift-left operation. A variable T stores the number of such shift-left operations. For $WF_1 = 31$, the value of T lies between 0 to 30 and, therefore, a 5-bit counter can be used. The greatest magnitude of $|-T\ln(2)|$ is 20.79, which implies that the result of the algorithm can be expressed in $Q\langle 32, 26\rangle$ format. The

Algorithm 4 Calculating $\ln(u_1)$, $u_1 \in [0.5, 1)$

```
T = 0;
while (u1[WF1 − 1] ≠ 1) do
  u1 ≪ 1; T++;
end while
y = 0; w = 1 − u1;
for (j = 0; j ≤ WL1; j++) do
  if (w ≥ 0) then
    s = 1;
  else if (w ≥ −0.75) then
    s = 0;
  else
    s = −1;
  end if
  if (j > WL1/2) then
    L = s2^−j;
  else
    L = ln(1 + s2^−j); // The right hand side is read from a table
  end if
  y = y − L;
  w = 2(w − s + s w 2^−j);
end for
return y − T ln 2; // T ln 2 are read from a table
```

precisions of the w and y variables can be set to use $\mathcal{Q}\langle 32, 28\rangle$ and $\mathcal{Q}\langle 32, 30\rangle$, respectively. Since the $\ln(u_1)$ result is a negative number in $\mathcal{Q}\langle 32, 26\rangle$ format, the value of $-2\ln(u_1)$ can be in $\mathcal{Q}\langle 32, 25\rangle$ format. However, since $-2\ln(u_1)$ is always a positive number less than 49, the input to the square root operation can be taken to be an unsigned number in the $\mathcal{Q}\langle 32, 26\rangle$ format. The $\ln(u_1)$ implementation utilizes 246 slices in an XC2V4000-6 FPGA and operates at up to 132 MHz.

To speed up the $\ln(u_1)$ calculation, one BlockRAM can be utilized to store some pre-defined values required by Algorithm 4. Here 30 values of $-T\ln(2)$ are precomputed and stored in a memory. Thirty-four different values of $-L = -\ln(1 + s2^{-j})$, for $0 \leq j \leq 16$ and $s = \{-1, 1\}$, are stored in the same BlockRAM. Also, to calculate $L = s2^{-j}$, for $16 < j \leq 32$, 32 values of $-L$ are stored in the memory for $s = \{-1, 1\}$. Altogether, 96 words of BlockRAM are used with the BlockRAM configured into a 512×36 aspect ratio. Since rescaling of the final result — and therefore the look-up of the value of $-T\ln(2)$ — takes place in the last clock cycle, there is no need to configure the BlockRAM into dual-port mode.

Since the calculation of $\ln(u_1)$ takes $WL_1 + 1$ clock cycles and the input to the square root module comes in sequence from the $\ln(u_1)$ module, the square root can be performed

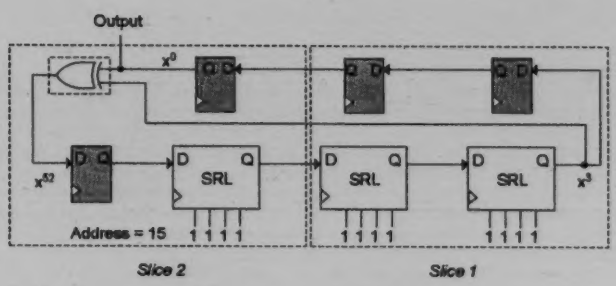

Figure 3.6: 52-bit LFSR design.

in a second pipeline stage while the $\ln(u_1)$ module calculates the next logarithm. Therefore, rather than using a multiplicative approach, a digit recurrence is utilized to reduce the area and to balance the pipeline delay between the two stages of the computation. The implemented square root circuit utilizes 173 slices in an XC2V4000-6 and runs at up to 148 MHz.

It is important to note that only one single-bit output LFSR is required to implement the PNG. This is due to the fact that the calculation of $f(u_1)$ takes $WL_1 + 1$ clock cycles, which is greater than the WF_1 clock cycles required to generate u_1. Therefore, using a LF-LFSR for PNG would not increase the overall noise generation rate. Similarly, to generate u_2, another 52-bit LFSR with single-bit output is required to be able to generate u_2. Conventionally, flip-flops would be used as storage elements to implement an LFSR. This is also the case when implementing a LF-LFSR since the output of flip-flops are required to be accessed. With two flip-flops in each slices, an n-bit LFSR will take up at least $\lceil n/2 \rceil$ slices. However, a four-input lookup table (LUT) in recent FPGAs can also function as a 16-bit shift register lookup table (SRL), with a single output accessed by the LUT's address lines [89]. This output allows the cascading of any number of 16-bit SRLs to create shift registers of arbitrary size. The SRL-based implementation can significantly decrease the area of the GVG. Figure 3.6 shows how a 52-bit LFSR, defined by the characteristic polynomial $p(x) = x^{52} + x^3 + 1$, can be implemented efficiently using four 4-input LUTs and four flip-flops to generate a single-bit PN per clock cycle. The input addresses of the three 16-bit SRLs in Figure 3.6 are all set to 15 in order to cascade them. A length of 52 bits was chosen to avoid pattern repetition for a large number of PNs. The implemented LFSR utilizes only two slices (in the same CLB) and its maximum operating clock frequency is 290 MHz.

The implemented GVG on the Xilinx Virtex-II XC2V4000-6 FPGA uses 3% of the con-

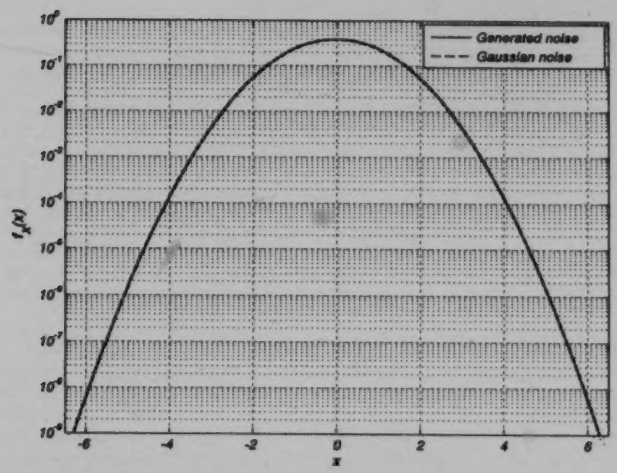

Figure 3.7: Gaussian PDF compared with PDF generated noise samples.

figurable slices, two BlockRAMs, and operates at 132 MHz, and hence generates $264/WL_1$ million noise samples per second. Figure 3.7 shows the PDF of 10^{13} generated noise samples. Even though Figure 3.7 shows that the PDF of the generated noise lies accurately over the normal distribution PDF within a $\pm 6.55\sigma$ interval about the zero mean, the main drawback of this technique is the relatively slow GV generation rate. The execution times of the iterative logarithm and square root algorithms are linearly proportional to the precision of u_1 [72]. Consequently, for the high precision PNs that are required to generate samples well into the tails of the Gaussian distribution, the noise generation rate will be relatively slow [76]. One effective solution would be to utilize high radix computation; however, the resulting hardware complexity would increase substantially [72, 105].

3.4.2 A GVG Using Non-Uniform Quantization and Table Lookup

A straightforward scheme for obtaining discrete values of $f(\cdot)$ is to store precomputed quantized values of $f(u_1)$, where $u_1 \in [0, 1)$ and $f(u_1) \in [0, \infty)$. As shown in Figure 3.8, $f(u_1)$ increases exponentially as u_1 approaches zero. Thus, a simple uniform quantization of $f(u_1)$ consumes a prohibitively large amount of memory. Alternatively, a *nonuniform quantization* version of $f(u_1)$ over the interval (0,1) can be stored in a smaller memory. An L-stage nonuniform quantization of $f(u_1)$ can be calculated by a recursive partition of the $(0, 1)$ interval. As shown in Figure 3.9, first the interval (0,1) is divided into 2^{q_1} segments and the function values within $(1/2^{q_1}, 1)$ are stored in a $2^{q_1} \times WL_1$-bit BlockRAM.

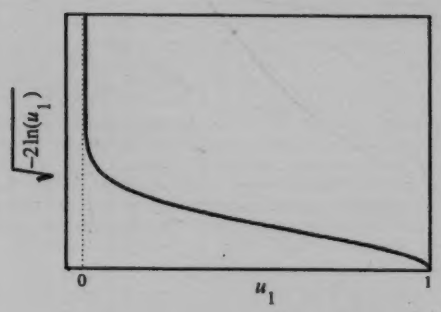

Figure 3.8: Plot of $f(u_1) = \sqrt{-2\ln(u_1)}$.

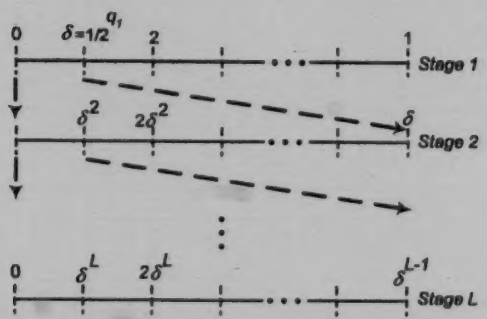

Figure 3.9: Non-uniform quantization of $(0, 1)$.

Then the interval $(0, 1/2^{q_1}]$ is subdivided into 2^{q_1} segments and the corresponding function values are stored in another BlockRAM. This procedure of nonuniform quantization is repeated recursively. The quantized values of $f(\cdot)$ are thus stored in $1 \leq L \leq \lceil WL_1/q_1 \rceil$ dedicated BlockRAMs. Clearly the smaller the quantization step, the greater the memory requirements for storing the quantized values. The number of stages, L, can be chosen so as to give the most accurate noise samples given the available precision of u_1, the memory aspect ratio and the available number of BlockRAMs.

The datapath of the GVG is shown in Figure 3.10. The PNG block uses a 52-bit LF-LFSR defined by the characteristic polynomial $p(x) = x^{52} + x^3 + 1$ to generate one $(L \times q_1) + q_2$-bit PN, every clock cycle. According to the nonuniform quantization of $f(\cdot)$, to obtain the $f(\cdot)$ value, the PNG block generates L pseudo-random numbers, each q_1 bits, to address the L BlockRAMs. If the first set of q_1 bits are not all zero, then the first $f(\cdot)$ BlockRAM is addressed. Otherwise, the second set of q_1 bits are checked. This addressing scheme is repeated until either q_1 is nonzero or the ℓ-th iteration is reached, where $\ell = 1, \ldots, L$. Note that if $L \times q_1$-bit zeros are generated in a row, then since $\ln(0)$

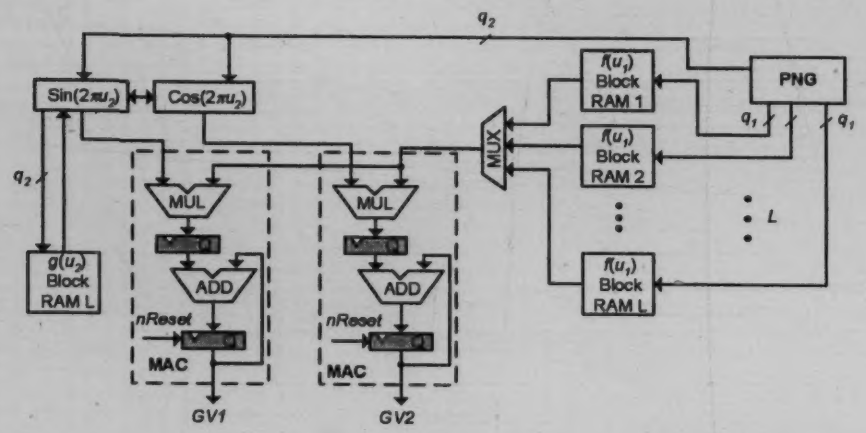

Figure 3.10: GVG datapath using non-uniform quantization and table lookup.

tends to infinity, the last memory location of the L-th $f(\cdot)$ BlockRAM is addressed.

The computational core of the GVG is two pipelined MACs, which are enclosed in dashed boxes in Figure 3.10. According to the CLT, if $x_i \sim \mathcal{N}(0,1)$ then $\sum_i x_i/\sqrt{K}$, as i approaches infinity, tends to be normally distributed. An accumulator can then be used to reduce the quantization error and smooth out the fluctuations of the obtained distribution. The accumulator is reset with *"nReset"* after summing K noise samples. The number K is selected to allow $1/\sqrt{K}$ to be performed using only shift-right operations (e.g., $K = 4, 16$, etc.). Note that by choosing $K = 2$, we cancel out with $\sqrt{2}$ factor in the $f(\cdot)$ equation. To calculate $g(\cdot)$, a uniform quantization version of $g_1(\cdot)$ over the interval $[0, 0.25]$ is stored in a BlockRAM. The q_2-bit PN u_2 is used to address the $g_1(\cdot)$ BlockRAM. Then the $\sin(2\pi u_2)$ and $\cos(2\pi u_2)$ blocks calculate $g_1(\cdot)$ and $g_2(\cdot)$ over the interval $[0, 1]$, respectively.

The parameters of the GVG design include the number of quantization stages L, the bit precisions of u_1 and u_2, the aspect ratios of the BlockRAMs and the number of accumulations. The noise generation rate depends on the number K of accumulations. The GVG quality depends strongly on the precision of u_1 and the number L of quantization stages. These parameters can be selected based on the desired GVG objectives. The effects of different parameter values were simulated using a compiled C model for a large number of samples and then optimized according to the resource configuration on the FPGA to achieve an efficient GVG with respect to the resource utilization, noise generation rate and PDF quality.

The GVG uses 5 BlockRAMs, only 1% of the slices in a Xilinx Virtex-II XC2V4000-6

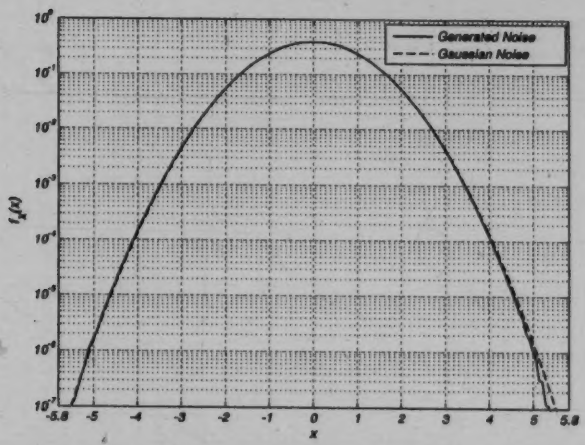

Figure 3.11: Ideal Gaussian and generated PDFs plotted on a logarithmic scale.

FPGA and operates at 165 MHz to generate $330/K$ million GVs per second. The $\sin(\cdot)$ and $\cos(\cdot)$ modules in Figure 3.10 require only 36 slices. Figure 3.11 plots the PDF of the theoretical Gaussian distribution together with the PDF of 10^9 generated GVs. The simulation parameters are $q_1 = 9$, $WL_1 = 32$, and $L = 4$. The four $f(\cdot)$ BlockRAMs are configured with a $2^9 \times 32$ aspect ratio. Other parameter settings include $q_2 = 10$, $WL_2 = 16$; consequently, the $g_1(\cdot)$ BlockRAM is configured with a $1K \times 16$ aspect ratio. Figure 3.11 shows that the PDF of the generated noise samples fits closely over the ideal PDF within only up to $\pm 5.0\sigma$. The primary reason is that the proposed simple nonuniform quantization is only applied to the domain of u_1 close to zero. The nonlinear region close to $u_1 = 1$ is not segmented as efficiently by the binary subdivision scheme as the domain gets close to $u_1 = 0$. For a higher quality GVG, the number of quantization stages must also be relatively larger which implies that a larger on-chip memory would be required.

3.4.3 Implementation of a GVG Using Piecewise Polynomial Curve Fitting

The non-uniform quantization and table lookup scheme suffers from low PDF accuracy especially at the tails of the distribution. The iterative approach can provide arbitrarily improved GVG quality; however, due to the iterative procedure of the convergent algorithms for the $\ln(\cdot)$ and square root operations, the GV generation rate is relatively slow when accuracy into the Gaussian tails is important. To avoid these limitations, a polynomial curve fitting approach can be applied to approximate $f(u_1)$ between $(0, 1)$ with high accuracy and relatively modest hardware requirements, while achieving a high GV generation

rate. In this application, a polynomial approximation involves representing the continuous function $f(u_1)$ with a polynomial $p(u_1) = \sum_{i=0}^{v} a_i u_1^v$ of finite degree v over an interval $[a, b)$. Various common curve fitting algorithms, such as linear or cubic interpolations or rational polynomial interpolations, differ from each other with respect to the amount of computational requirements and the residual error, which is given by $\xi = \|f(u_1) - p(u_1)\|$ where $\|\cdot\|$ denotes a suitable norm. For example, the least-squares polynomial regression $\xi^2 = \sum \left[f(u_1) - p(u_1)\right]^2$ can be calculated based on the vertical offset between the $f(u_1)$ and $p(u_1)$ curves or based on the perpendicular offset. A maximum likelihood estimate of the polynomial coefficients can be obtained using the orthogonal least squares fit (OLSF) method [108], which minimizes the summed square of the residuals $\sum_i d_i^2$, where d_i denotes the maximum perpendicular distance from the point on the polynomial approximation to the point on $f(u_1)$. Note that in addition to the error due to the approximation of $f(u_1)$, there is another source of error called *round-off error* or *quantization error* due to the finite wordlength. Truncation errors and rounding errors can be considered as subsets of quantization errors.

Due to the nonlinear shape of $f(\cdot)$, a relatively high-degree single polynomial is required to approximate $f(\cdot)$ accurately over the interval $(0, 1)$. The polynomial degree has a direct effect on the residual approximation error, hardware complexity, latency and resource utilization. In general, the common weighted sum of powers form of $p(\cdot)$ requires $v(v+1)/2$ multiplications and v additions to evaluate $p(\cdot)$. The "factored" representation $p(u_1) = a_0 + (a_1 + \cdots (a_{n-1} + a_n u_1)u_1)u_1 \cdots)u_1$ reduces the number of multiplications down to v and still requires v additions. A more compact implementation could be obtained using time-shared hardware, but the resulting calculation time would be increased. Instead, to increase the speed of the computation, the $(0, 1)$ interval can be divided into k smaller segments separated by a set of points called *joints*. The fewer the segments, the higher the degree of the polynomial that is required to approximate $f(\cdot)$ within each segment. Over each interval, a (different) polynomial of lower degree can provide a similarly accurate approximation to $f(\cdot)$. Even though a piecewise function $f(\cdot)$ can be approximated with a lower degree polynomial in each segments, discontinuities can arise at the ends of the intervals used for separate definitions. On the other hand, although any choice of segment boundaries can easily be implemented in a software simulation, some choices imply overly complex decoding circuitry and are thus undesirable for high-speed GVG realization. Therefore, appropriate segmentation is crucial to the accuracy and the speed of

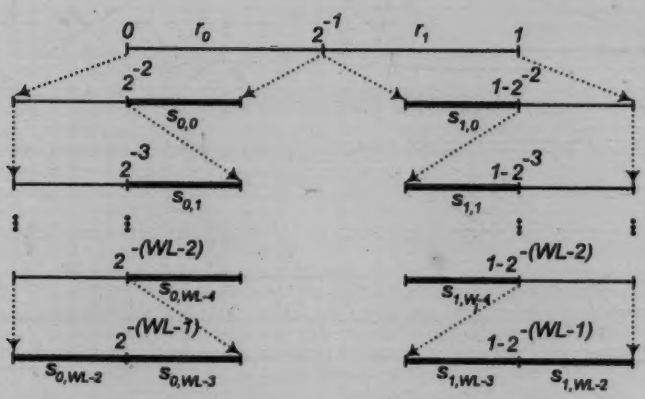

Figure 3.12: Segmentation of $u_1 \in (0,1)$.

the resulting GVG.

As shown in Figure 3.8, the function $f(u_1)$ has two high-slope regions: in the vicinity of $u_1 = 0$ and close to $u_1 = 1$. Since a small input change may lead to a (very) large output change, the input domains near 0 and 1 need smaller segments than the relatively linear regions in the middle of the domain. Different nonuniform segmentation schemes were already proposed in [61] and [62]. The segmentation scheme in [61] utilizes on-chip memory to store nonuniformly quantized values of $f(\cdot)$. In [61], only the nonlinear region close to 0 is segmented nonuniformly. In this region the precomputed values of $f(\cdot)$ are stored in memory. The method in [62] uses nonuniform segmentation in both the regions close to $u_1 = 0$ and $u_1 = 1$. It also uses a piecewise linear approximation for more accurately computing $f(\cdot)$ within each segment. The CLT [14] was exploited in both [61] and [62] to improve the statistics of the resulting distribution by averaging multiple GVs.

We use a hierarchical segmentation method: the domain $(0,1)$ of u_1 is divided into two subintervals, $r_0 \in (0, 0.5)$ and $r_1 \in [0.5, 1)$. Let $u_1 \in [0,1)$ be represented as an unsigned fixed-point number with WL_1 bits of precision. The value of u_1 is $\sum_{i=1}^{WL} 2^{-i} u_{1_i}$ and its bit structure can be denoted as $u_{1_{WL_1-1}} \cdots u_{1_0}$. The value, 0 or 1, of the MSB bit of u_1 indicates whether a particular u_1 resides in subinterval r_0 or r_1, respectively. The subinterval r_0 is segmented logarithmically into $WL_1 - 1$ segments from $u_1 = 0.5$ down to 0, as shown in Figure 3.12. Subinterval r_1 is segmented similarly from $u_1 = 0.5$ up to 1. Each segment is denoted by $s_{h,w}$, where binary subscript h specifies the half range (r_0 or r_1), and w denotes the segment number $w = 0, \cdots, WL_1 - 2$. Each segment is then subdivided uniformly into $M = 2^m$ segments. One can verify that the total number of

segments is $2^{m+1}(WL_1 - 1)$. The value of $f(\cdot)$ within each segment is then approximated using a polynomial $p(\cdot)$. A piecewise continuous (differentiable) scheme is utilized where the ends of intervals are smooth. Therefore, the polynomial pieces combine smoothly. Any such smooth piecewise polynomial function is called a *spline*. We have used the MATLAB function "polyfit(x,y,n)" to find the coefficients of a polynomial $p(\cdot)$ of degree n that fits the polynomial $p(\cdot)$ to $f(\cdot)$, in a least squares sense. The gradient weights (coefficients) of the polynomials for approximating $f(u_1)$ within each segment are then optimized based on the OLSF method to minimize the residual error.

As u_1 approaches 2^{-WL_1} from above (for the nonlinear region just above $u_1 = 0$) or when u_1 approaches $1-2^{-WL_1}$ from below (for the nonlinear region just below $u_1 = 1$), the slope $\partial f(u_1)/\partial u_1$ of $f(u_1)$ tends toward infinity and, therefore, the coefficients of $p(u_1)$ become large. However, the value of $p(u_1)$ lies within $(0, \sqrt{2WL_1 \ln 2})$. For example, for a $WL_1 = 32$-bit representation of u_1, the value of $p(u_1)$ lies in $(0, 6.66)$ and can be represented accurately in 16-bit fixed-point format. Storing the large coefficient values of $p(u_1)$ on-chip requires large memories, increases the hardware complexity and slows down the variate generation rate. To overcome this problem, [62] proposed to store scaling factors (multiples of two) along with the coefficients into an on-chip memory to reduce the magnitude of the slope, trading off precision for range. Instead we use a scaling scheme that stores only adjusted coefficients of $p(u_1)$ in an on-chip memory. This scheme reduces the memory requirements, decreases the hardware complexity, maintains the accuracy of the computation, and does not sacrifice precision for range as in [62]. However, a simple circuit is required to scale and thus accurately represent the input value u_1.

For clarity, we explain the scaling method assuming that each segment is approximated using a (piecewise) linear polynomial $p(u_1) = au_1 + b$. The scaling scheme is independent of the number of segments (precision of u_1) and the order of polynomial. Assume that the PNG generates an unsigned uniformly distributed number u_1. When u_1 lies within r_0, u_1 is shifted left until the most significant bit (MSB) bit is 1. Thus if u_1 lies in segment $s_{0,w}$, a new scaled variable $\tilde{u}_{1,w}$ is defined as $\tilde{u}_{1,w} = 2^w u_1$. To compensate for the scaling of u_1, the $a_{0,w}$ slope of segment $s_{0,w}$ is shifted to the right w bit positions as $\tilde{a}_{0,w} = 2^{-w} a_{0,w}$. Thus the largest slope values are scaled down by a factor of $\lambda_d = 2^{-WL_1+2}$ and stored in memory. Hence we compute

$$p(\tilde{u}_1) = \tilde{a}\tilde{u}_1 + b = 2^w \tilde{a} u_1 + b. \tag{3.4}$$

When u_1 lies within $s_{1,w}$, to scale the slopes and intercepts of $p(\cdot)$ accurately, the variable u_1 is transformed to a new variable $\hat{u}_1 = 1 - u_1$. Thus as $u_1 \rightarrow 1^-$, $\hat{u}_1 \rightarrow 0^-$. Similarly, when u_1 resides in segment $s_{1,w}$, with $\tilde{u}_1 = 2^w \hat{u}_1$, $p(\tilde{u}_1)$ can be written as

$$p(\tilde{u}_1) = -2^{-w} a \tilde{u}_1 + (a + b) \tag{3.5}$$

The scaled slope and intercept of the polynomials of each segment can in this way be accurately stored in an on-chip memory. According to the value of u_1, a small scaling circuit provides the subinterval r_i, the segment number $w = 1, \cdots, WL_1 - 1$, the subsegment number $m = 1, \cdots, M$, and the scaled value of u_1. Then the scaled coefficients of $p(u_1)$ can be addressed and read directly from memory to compute (3.4) or (3.5) as an accurate approximation to $f(u_1)$.

Figure 3.13 shows a dataflow diagram illustrating the evaluation of $p(\tilde{u}_1)$ in its factored form and the corresponding hardware datapath. The approximated coefficients can be stored in an on-chip memory. One important hardware constraint that should be considered is the amount of on-chip memory in the FPGA. If the FPGA has ϑ bits of memory and each coefficient is represented using c bits, then $\lfloor \vartheta / c \rfloor$ is the maximum number of coefficients that can be stored on-chip. One can thus trade off the order of the polynomial with the number of segments (limited by the on-chip memory). To generate one Gaussian sample, $f(u_1) = p(\tilde{u})$ is multiplied by $g_1(u_2) = \sin(2\pi u_2)$. The core of the GVG contains pipelined fixed-point multipliers, adders, registers and routing resources. The operations are pipelined and scheduled to maximize the output rate. As discussed in Section 3.2, we used combined generators with different periods of $\rho \approx 2^{88}$, $\rho \approx 2^{113}$, and $\rho \approx 2^{258}$ to produce 32-bit and 64-bit PNs every clock cycle.

The addressing unit (AU) calculates the scaled values of u_1, namely $\tilde{u}_1$, identifies the half range h (the MSB bit of u_1), the segment number w, and the subsegment number m. To determine the segment number w of a given PN input u_1, the AU uses a small leading one detector (LOD) circuit [109]. Since the AU is in the critical path of the GVG, its operating rate limits the output rate of the GVG. The tree structure of the LOD for an 8-bit u_1 is shown in Figure 3.14. The string of WL_1 bits generated by the PNG is first partitioned into $WL_1/2$ pairs of adjacent bits. For each pair, a 2-bit leading zero count is generated. The high order bit also indicates if a 1 is detected in the string. At the next level, the results for adjacent pairs are combined, a multiplexer selects the count from one of the pairs, and new low order bits are appended to the count. This scheme is repeated

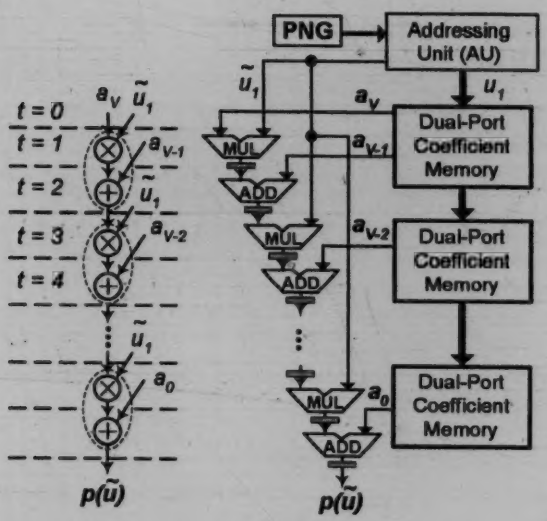

Figure 3.13: The datapath for calculating $f(\cdot)$ function.

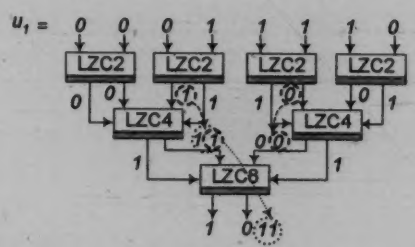

Figure 3.14: The tree structure of leading zeros counting.

for all $\log_2(WL_1)$ levels. Some speed-up can be obtained by using 4-bit or 8-bit groups and by using larger multiplexors. In our implementation, the AU is pipelined with three stages to achieve a high GV generation rate. Another approach to detect the leading one index is proposed in [110]. As shown in Figure 3.15, the 2-bit LOD generates a valid bit v and a position bit p that can be extended for a 4-bit LOD. Two 4-bit LODs are utilized to generate an 8-bit LOD. The generated triple (r, w, m) from the AU is then used to address the coefficient memory. For $WL_1 = 32$ and $M = 8$ ($WL_1 - 1$ segments in each interval, and M subsegments for each segment), only one BlockRAM is required to store the $2 \times 2 \times 31 \times 2^3 = 992$ coefficients of 62 polynomials. Since the polynomial coefficients are accessed simultaneously, the BlockRAM must be configured in dual-port mode.

Before realizing the GVG on an FPGA, the parametric model of the datapath was simulated and verified. Table 3.5 summarizes the implementation results for the new GVG on

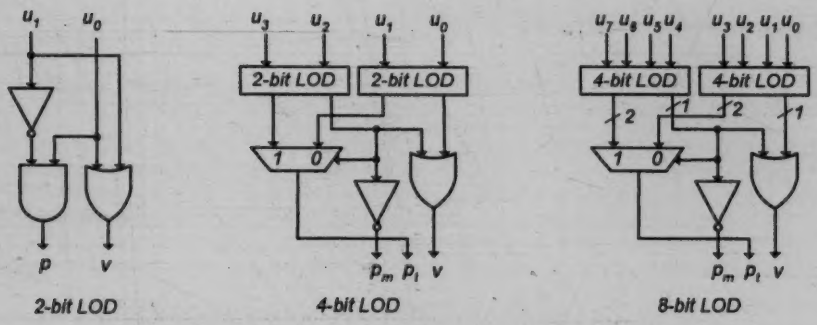

Figure 3.15: Logic diagram for leading one detector.

Table 3.5: Typical realizations of the new GVG.

Device	I[a]	II	III	IV
Bitwidth of u_1	32	64	32	32
Period of PNG	$\approx 2^{88}$	$\approx 2^{258}$	$\approx 2^{88}$	$\approx 2^{133}$
Max. deviation	6.66σ	9.41σ	6.66σ	6.66σ
Clock freq. (MHz)	253.52	253.16	107.74	221.68
Output rate (MGVs/sec)	506	506	214	442
Number of slices	344	441	343	705
Resource utilization	1.4%	1.9%	11%	1%
On-chip memory blocks	2	2	2	2

[a]Designs I and II were synthesized for a Xilinx Virtex-II XC2V4000-6 FPGA. Design III was implemented for a Xilinx Spartan-IIE XC2S300E-7 FPGA. Design IV was synthesized for an Altera Stratix EP1S80F1508C6 FPGA. The latency of all four GVGs is 10 clock cycles.

four different FPGAs, utilizing linear curve fitting. One important feature of the proposed design is that, due to the efficiency of the scaling scheme, the datapath of the GVG can be implemented in 16-bit fixed-point format, independent of the precision of u_1. In fact, u_1 can be scaled successfully to generate GVs with various tail accuracies using the same 16-bit datapath. To do this, only the PNG and AU must be modified slightly. Extensive test analysis verified that the proposed 16-bit design preserves the computation accuracy. Fortunately, the 16-bit GV format tends to be a common precision for many DSP applications. The Xilinx AWGN core [75] also uses the 16-bit fixed-point format. One important point is that the datapath of the new GVG is conveniently scalable to permit faster GV generation. If there are sufficient resources available on the FPGA beyond those required by a single GVG datapath, then multiple instances of the same GVG datapath, with different initial seeds for the PNGs, could be readily instantiated to speed up the total GV generation rate.

Figure 3.16 shows the layout of the $0.126\,mm^2$ GVG chip designed in a 90-nm CMOS technology using a dual-threshold standard cell library. The core area is dominated by the

Figure 3.16: The 0.126 mm^2 GVG chip layout in 90-nm CMOS technology.

dual-port coefficient ROM and the single-port sine ROM. Without access to a commercial read-only memory (ROM) core generator, we implemented the ROM array using standard cells. Custom ROM blocks would significantly reduce the area. The unlabeled area in the core layout is occupied by two-stage pipelined multipliers, adders, registers and routing resources. The core operates at 537 MHz, generating more than one billion GVs per second while dissipating 12.3 mW of dynamic power. The static power dissipation is estimated to be 9.91 mW.

3.5 The GVG Statistical Tests

Normality tests are well-known statistical measures used to determine if generated GVs fit a standard normal distribution [111]. They are based on various key characteristics of the normal distribution. In this section we evaluate the normality of the variates produced by the GVG realizations specified in Table 3.5.

The power of statistical tests differs depending on the nature of any deviations from ideal normality, such as skewness or an otherwise inaccurate distribution. Normality tests are performed either graphically or numerically. Figure 3.17 superimposes the PDF of 10^{11} generated GVs on top of a PDF plot of the theoretical normal distribution. The two plots are indistinguishable over $\pm 6.0\sigma$ from visual inspection. To generate the GVs at the tails of the distribution, where $|n| > 6.0\sigma$, at least 10^{13} samples are required to produce the PDF, which takes a prohibitively long time. Instead, the PDF of the Gaussian variable n, $f_N(n)$,

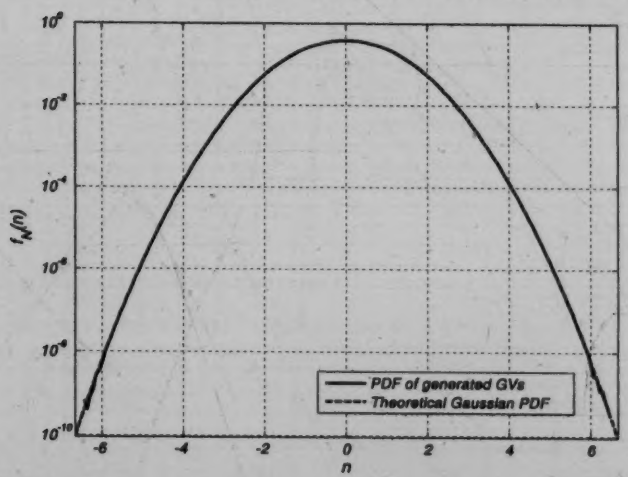

Figure 3.17: Gaussian PDF compared with the PDF 10^{11} generated GVs.

can be expressed in terms of its CDF, $F_N(n)$. Note that the PDF can be written as

$$f_N(n) = \frac{d}{dn} F_N(n) = \frac{d}{dn} \Pr(N < n). \tag{3.6}$$

The "importance sampling" expression can be written using Bayes' law [14] as

$$F_N(n) = F_N(n|U_1 < u_\tau) \Pr(U_1 < u_\tau) + F_N(n|U_1 \geq u_\tau) \Pr(U_1 \geq u_\tau). \tag{3.7}$$

To measure the PDF in the tails of the distribution, GVs such that $U_1 < u_\tau \ll 1$ are first generated. The PDF of the generated GVs can then be given as

$$f_N(n) = \frac{d}{dn} F_N(n|U_1 < u_\tau) \Pr(U_1 < u_\tau), \quad \forall \ |n| > \sqrt{-2\ln(u_\tau)}. \tag{3.8}$$

In this method, we do not generate PNs close to 1 and, therefore, the second term of Equation (3.7) approaches 0. Using this method, instead of generating 10^{13} GVs, the PDF at the very ends of the distribution tails can be assessed using only 10^9 GVs. From a resulting PDF plot one might reject the claim of normality if the distribution deviates significantly from a bell-shaped normal distribution. Similarly, the CDF plot can also be used to judge the symmetry and skewness of the generated distribution. However, it is usually difficult to assess the accuracy visually, particularly at the tails of the distribution.

It is usually more convenient to compare two linear functions. The Quantile-Quantile (Q-Q) plot of the generated GVs is shown in Figure 3.18. If the points in the plot of $\mathrm{G}^{-1}(F(n))$ versus n lie roughly on a straight line with intercept $\mu = 0$ and slope $\sigma = 1$, then one might conclude that n is normally distributed [111]. A departure from the expected

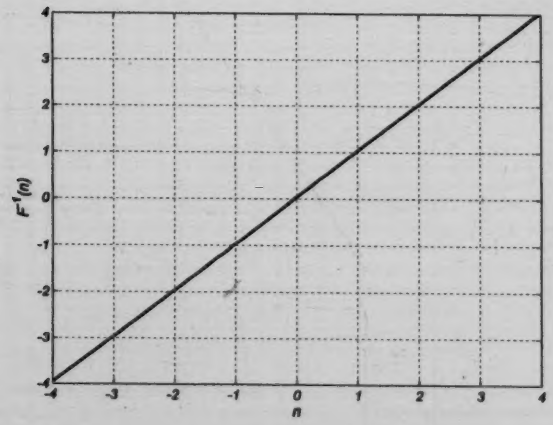

Figure 3.18: Inverse CDF of the generated GVs.

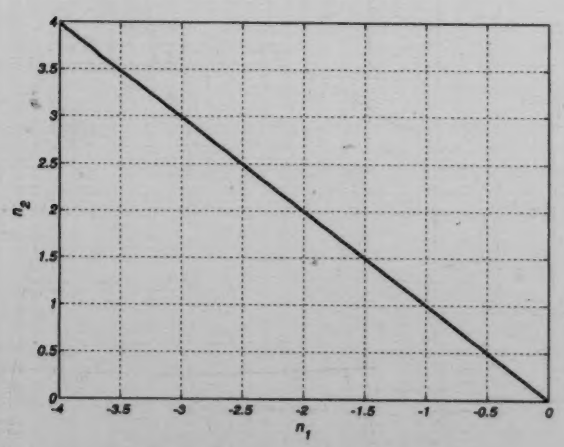

Figure 3.19: Plot of n_1 versus n_2.

line indicates departure from normality. Specifically, an S shaped-curve indicates lighter than normal tails. The generated variates do indeed pass this test.

Deviations from normality can also be judged using a scatter diagram [111], such as the one shown in Figure 3.19. The scatter diagram plots the upper half of the ordered generated GVs against the lower half. A negative unit slope indicates symmetry, a negative slope exceeding unity in absolute value indicates positive skewness, and a negative slope less than unity in absolute value indicates negative skewness [111]. Analytically, the standard third moment ($\sqrt{\beta_1}$) and fourth moment (β_2) of the generated distribution can be evaluated as

$$\sqrt{\beta_1} = \frac{\mathrm{E}(N-\mu)^3}{\{\mathrm{E}(N-\mu)^2\}^{3/2}} \tag{3.9}$$

and

$$\beta_2 = \frac{\mathrm{E}(N-\mu)^4}{\{\mathrm{E}(N-\mu)^2\}^2} \tag{3.10}$$

to characterize the skewness and kurtosis of the generated PDF, respectively [111]. For a normal distribution $\sqrt{\beta_1} = 0$ and $\beta_2 = 3$. If $\sqrt{\beta_1} < 0$ then the distribution is skewed to the left, and $\sqrt{\beta_1} > 0$ means it is skewed to the right. If $\sqrt{\beta_1} = 0$ and $\sqrt{\beta_2} < 3$ then the distribution is symmetric but the tails are thinner than normal, while $\sqrt{\beta_2} > 3$ indicates that tails are thicker than normal.

Relying solely on graphical goodness-of-fit techniques can lead to false conclusions [111]. A detailed graphical analysis should always be supported by formal hypothesis testing. The composite null hypothesis $\mathcal{H}_o : n \sim \mathcal{N}(0,1)$ asserts that the generated variates follow the standard normal distribution. The Pearson Chi Square χ^2_α test can be utilized to determine the validity of $\mathcal{H}_o$ with a desired significance level $\alpha = 0.05$ [111]. The χ^2_α test involves quantizing the horizontal axis of the PDF into γ equiprobable cells. We have chosen the number of cells to ensure that at least five variates reside in each cell. The $\chi^2_{\alpha,\gamma-1}$ statistic can be calculated based on the actual and expected number of samples appearing in each cell and serves as an overall quality metric as follows,

$$\chi^2_{\alpha,\gamma-1} = \sum_{i=1}^{\gamma} \frac{(m_i - KP_i)^2}{KP_i} \tag{3.11}$$

where K is the number of observations. For each cell i, m_i and P_i are the number of variates and the probability that each variate falls into cell i, respectively. A normal distribution is completely specified by two measures, the mean and the standard deviation. Since these measures can be computed from the data, the number of degrees of freedom (dof) is reduced by two (the number of parameters computed). So the number of dof is reduced from $\gamma - 1$ to $\gamma - 3$ [18]. Since the measured $\chi^2_{\alpha,\gamma-3}$ for our GVG was less than the threshold value, we accept $\mathcal{H}_o$.

In general, the PDF based χ^2_α test is not an especially powerful test for normality [111]. A weakness of χ^2_α test is the arbitrariness of the choice of cells. The Anderson-Darling statistic $\mathcal{A}^2$ measures the integrated quadratic deviation between the empirical distribution function $F(n)$ and the theoretical function $F_N(n)$, multiplied by a weighting function $\psi(n)$ as follows,

$$\mathcal{A}^2 = \int_{-\infty}^{\infty} [F_N(n) - F(n)]^2 \psi(n) dn \tag{3.12}$$

where $\psi(n) = 1/F(n)(1 - F(n))$. The weighting function $\psi(n)$ is used to enhance the

sensitivity of the statistic in the distribution tails. The $\mathcal{A}^2$ test statistic for a normal distribution can be calculated numerically as [111]

$$\mathcal{A}^2 = -K - 1/K \sum_{i=1}^{K} (2i-1)[\ln(F_Z(z_i) + \ln(1 - F_Z(z_{K+1-i})] \qquad (3.13)$$

where $F_Z(\cdot)$ is the standard normal CDF and $\{z_i\}$ are the ordered generated GVs. Since the measured parameter of $\mathcal{A}^2(1 + 0.75/K + 2.25/K^2)$ was less than the critical value of 0.752, we can again accept $\mathcal{H}_o$ for our GVG.

The most important test is the correlation test when GVG is used in a MC simulation. Small correlations in the random number generator can easily lead to spurious effects and invalidate simulation results. It is important to note that the quality of the randomness of the GVG is dominated by the behaviour of the PNG. If the generated PNs are not truly independent, then "random" pairs (n_i, n_{i+1}) will lie on a spiral [112]. We considered several different PNG designs with different autocorrelation properties. It was concluded in Section 3.2 that the combined linear and nonlinear generators tended to have superior randomness properties. To further reduce regularities, one can additively combine a nonlinear PNG [96, 99] with one of the linear CTGs with large period such as the one with $\rho \approx 2^{258}$ proposed by L'Ecuyer [94] that passes most of the major statistical tests and various MC tests [81]. A nonlinear PNG can be implemented by combining several small nonlinear PNGs [99] and storing the samples in an on-chip memory. To verify the correlation among generated GVs, a sequence containing 10^7 variates generated by our GVG was subjected to the linear Pearson's correlation test [111] to estimate the correlation between random variates. No regular lattice structure can be observed visually, as shown in Figure 3.20. Figure 3.21 plots the autocorrelation values over the range of lags ± 2048 for 10^7 generated GVs.

3.6 Conclusions

This chapter presents three different approaches to implementing a GVG based on the Box-Muller algorithm. A fast and compact GVG was described that has a higher Gaussian sample generation with lower hardware cost than published designs. The functions required by the Box-Muller method were approximated using hierarchical segmentation and a novel recursive scaling scheme to maintain the approximation accuracy. Specifically, the proposed GVG design uses two levels of segmentations, a non-linear segmentation along with a uniform segmentation. The new scaling scheme avoids sacrificing precision for the range, as is

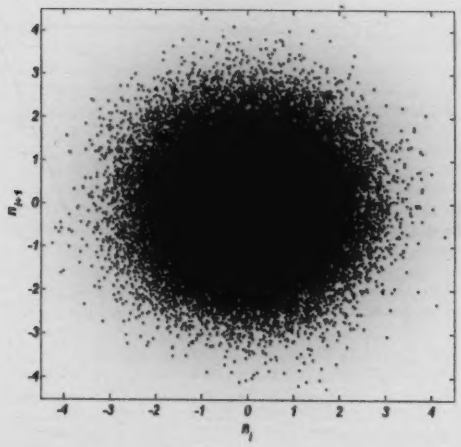

Figure 3.20: Statistical dependency of n_i and n_{i+1} for 10^7 generated GVs.

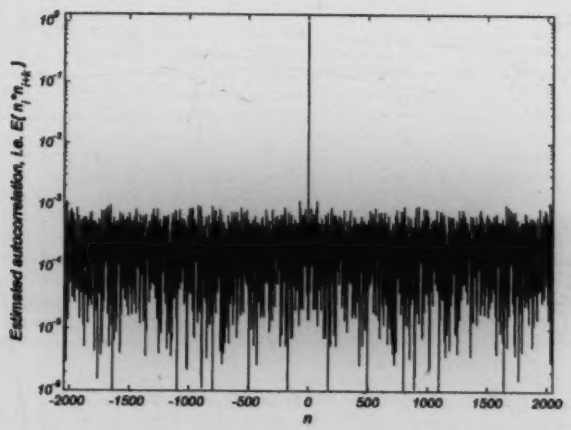

Figure 3.21: Autocorrelation among n_i and $n_{i\pm 2048}$ for 10^7 generated GVs.

the case in other published methods. Moreover, the same 16-bit GVG datapath can generate Gaussian variates with larger σ values, with a slight increase in delay and resource usage, as shown in Table 3.5. The statistical characteristics of the GVG were evaluated and confirmed using multiple standard statistical goodness-of-fit tests. The implementation costs and performance were illustrated by typical realizations for FPGAs and a 90-*nm* CMOS ASIC. Our proposed GVG uses only 1.4% of the Xilinx Virtex-II XC2V4000-6 FPGA and operates at 253 MHz, generating 506 million Gaussian variates per second within up to $\pm 6.66\sigma$. To accurately achieve a range of 9.41σ, without performance loss, only 1.9% of the same FPGA is required. This GVG should therefore be of significant assistance in the

characterization of low-BER systems.

The proposed designs are all parameterizable and, depending on the desired values of the GVG objectives, the parameters of the designs can be configured. Before realizing the proposed designs on an FPGA, the parametric models of the datapaths were all simulated and verified in Matlab and C. The datapaths are also conveniently scalable. If there are sufficient resources available on the FPGA beyond those required by the implemented design under test, the Gaussian sample generation can always be sped up by instantiating multiple instances of the GVG datapaths with different initial seeds for the PNGs.

Chapter 4

SOS-based Fading Channel Simulators

Wireless communication systems are being designed to operate over radio channels for a variety of environment and weather conditions. While it is possible to build prototypes of a proposed system and then field test them in different locations, such an approach will be quite expensive and will not provide useful feedback in the early stage of the system design when a number of candidate designs must be explored. Moreover, propagation conditions are almost impossible to repeat for the comparative analysis of simulation results. A more practical approach is to create appropriate models for the channel and then base the initial design on these models. Numerous wireless channel models have been proposed to characterize time and/or space-variant propagation environments [16, 23, 24, 113–115]. These channel models have led to different simulator designs that can be efficiently used in the development and accurate error-rate performance evaluation of wireless systems. A channel simulator should mimic the propagation characteristic faithfully since the accuracy of the performance estimation under real world conditions can make the difference between success and failure.

One of the widely used approaches to simulate fading channels are the SOS-based methods [116]. The basic idea behind SOS-based fading channel simulators is that when a sinusoidal carrier is transmitted and subjected to multipath fading, the received signal can be modeled as a superposition of waveforms received from different propagation paths. Since the nature and orientation of obstacles in the wireless channel are not known in advance, the received waveforms can be considered to be stochastic processes. In the SOS approach, the flat-fading process is modeled by superimposing sinusoidal waveforms with amplitudes, frequencies and phases that are selected appropriately to generate the desired

statistical properties.

Different software-based fading channel simulators based on the SOS approach have been proposed [117–121]. Programmers endeavor to map the parallel operations onto the very long instruction word (VLIW) architecture of high-performance DSPs and/or use the SIMD instructions of general-purpose processors to speed-up the simulation. Even using optimized software simulators running on high-performance processors, the speed is limited by the inherently sequential instruction execution and the lack of specialized functional units for MC simulation. In addition, the demanding performance requirements of wireless applications, along with the increasing computational complexity of baseband algorithms, have greatly increased software-based simulation loads. Therefore, the required run times for the accurate performance evaluation of the most recent low-BER baseband algorithms are becoming prohibitively long, which makes software-based simulation an inefficient technique.

New simulation techniques, such as importance sampling, have been proposed to increase the frequency of error events by introducing a bias into the statistical distribution of the random variables (such as noise) [122]. The main drawback of these techniques is that the statistical distribution of the channel effects should be known, and in most instances the channel has to be "reasonably linear". Moreover, these techniques may only be applicable for the simulation of some parts of the system while other parts still require conventional MC simulation.

Although it is much easier to design and implement a fading channel emulator in software than in hardware, hardware-based simulators have been shown to provide several orders of magnitude speed up in performance evaluation over software-based simulators [123, 124], significantly reducing the design time. Hardware-based fading channel simulators use digital hardware [119, 125–130] or employ analog techniques [117, 120, 125] for at least part of the baseband signal processing. Regardless of the selected interface (RF, analog baseband, or digital baseband), a digital fading channel simulator [128] is usually preferred to achieve the best possible accuracy, flexibility, and repeatability [123]. The majority of digital implementations use a general-purpose processor or DSPs [117–121]. Recent increases in the performance of FPGAs offer opportunities to reduce the cost and complexity required when implementing a channel simulator [131]. Some other implementations [123] use an FPGA combined with a DSP platform to implement the required computationally-intensive procedures of a channel model. Realizing the PL algorithm along

with the fading channel simulator on the same FPGA can simplify the required hardware and significantly speed up the evaluation process. Hence, digital implementations of multipath propagation models on rapid prototyping platforms (RPPs) are of great interest.

In order to implement a Rayleigh fading channel simulator on an FPGA, it is crucial to find a statistically-accurate SOS model that (1) can be efficiently mapped onto the regular architecture of the FPGA, and (2) provides the desired statistical properties of the fading channel. Despite the extensive acceptance and application of the SOS-based fading channel model, this model has limitations that should be studied and determined before software/hardware implementation. Among the different available algorithms for the generation of correlated Rayleigh random variates, as we show later, some do not produce statistically accurate fading variates and some are infeasible for hardware implementation.

The rest of the chapter is organized as follows. Section 4.1 reviews important fading channel parameters. These parameters are referenced in this chapter and also in Chapters 5 and 6. Two fundamental models for wireless channels are presented in Section 4.2. Section 4.3 gives an overview of important stochastic properties of radio channels. The literature on the modeling and analysis of multipath fading radio channels is vast. While a complete review of the literature is outside the scope of this chapter, a short but important review of the SOS-based modeling techniques is presented in Section 4.4. In Section 4.5, two compact implementations of the SOS-based fading simulator are described. The fading simulator uses only 1% of the widely-available Xilinx Virtex-II XC2V4000-6 FPGA device while generating over 200 million complex Rayleigh fading variates per second. The statistical properties of the generated fading variates are also evaluated. Section 4.6 presents a novel fading channel model based on the SOS approach. Using numerical simulation it is shown that the proposed model accurately reproduces the desired statistical properties of the fading envelope. The fixed-point fading channel simulator is designed and the accuracy, efficiency, and flexibility of the design are discussed. The discrete-time fading channel simulator is implemented on different FPGAs and the statistical properties of the generated fading variates are verified against the analytical channel model. A fixed-point implementation of the fading channel simulator on a FPGA utilizes 5% of the configurable resources and generates over 200 million 16-bit fading variates per second. The proposed digital channel simulator is compact enough to be integrated along with many communication circuits of likely interest. Finally, Section 4.7 makes some concluding remarks.

4.1 Parameters of Fading Channels

Generally, a multipath fading channel can be characterized statistically as a LTV system (filter) that models the superposition of multipath attenuated and delayed component signals at any instant of time [132]. We denote the low-pass impulse response of LTV channel as $c(t;\tau)$ and the corresponding time-varying frequency response as $C(t;f)$. While multipath propagation results in the spreading the transmitted signal in time, the time variations in the channel impulse response (or frequency response) result in the frequency spreading of the transmitted signal, generally called *Doppler spreading* as described in Chapter 2. Consequently, a multipath fading channel can be characterized as a doubly-spread channel in time (due to the reflected and scattered propagation paths) and frequency (due to the Doppler shift). The following channel parameters and fading conditions should be studied before discussing the modeling and simulation of multipath fading channels [15,18,19,115]:

• *Delay spread:* Maximum excess delay or *maximum* delay spread T_m is the delay between the first and the last component of the signal during which the received power falls below some threshold level, e.g., $\mathcal{X}$ dB below the strongest component. The *power delay profile* represents the average power associated with a given multipath delay, and is measured empirically. Since some channels with the same value of T_m can have different power delay profiles [19], a more useful parameter is the *root mean square* (rms) delay spread σ_τ defined as $\sigma_\tau = \sqrt{\langle\tau^2\rangle - \langle\tau\rangle^2}$ where

$$\langle\tau\rangle = \frac{\sum_{\ell=1}^{L} p_\ell \tau_\ell}{\sum_{\ell=1}^{L} p_\ell},$$

p_ℓ denotes the power coming along the ℓ-th propagation path, τ_ℓ is the time taken by the ℓ-th component, and $\langle\tau^2\rangle$ is the *mean square delay* given by

$$\langle\tau^2\rangle = \frac{\sum_{\ell=1}^{L} p_\ell \tau_\ell^2}{\sum_{\ell=1}^{L} p_\ell}.$$

The value of σ_τ is directly related to the minimum symbol period that can be used in order to avoid excessive ISI. The maximum delay spread is usually characterized by the rms delay spread, i.e. $\sigma_\tau = T_m$. If σ_τ is large, we expect to see considerable pulse broadening.

• *Channel coherence bandwidth:* Similar to the delay spread parameters in the time domain, the *coherence bandwidth*, B_c, is used to characterize a channel in the frequency domain [115]. The coherence bandwidth can be defined as a statistical measure of the range of frequencies over which the channel can be considered *flat* (i.e., the channel passes

all spectral components with approximately equal gain and linear phase). Thus, B_c is the range of frequencies over which all frequency component amplitudes are correlated. Consequently, the spectral components in that range fade together and two waveforms with frequency separation greater than B_c are affected quite differently by the channel. The coherence bandwidth is usually defined as the bandwidth (BW) over which the channel's transfer function has a correlation of at least 0.5 [115] as

$$B_c = \frac{0.276}{\sigma_\tau} \approx \frac{1}{5\sigma_\tau}.$$

Note that σ_τ and B_c are inversely proportional to one another, although their exact relationship is a function of the propagation environments. Thus it is possible to quantify the pulse broadening by the rms delay spread and/or the low-pass BW of the channel.

Delay spread and coherence BW are parameters that describe the *time dispersive* nature of the channel in the local area caused by multipath propagation. They do not offer information about the *frequency dispersive* nature of the channel caused by the relative motion between the receiver and transmitter. In small-scale region, the time varying nature can be described by the Doppler spread and coherence time parameters.

• *Doppler spread:* Provides a measure of how rapidly the channel impulse response varies in time. The larger the value of the Doppler spread B_d, the more rapidly the channel impulse response is changing with time. Doppler spread is also the range of frequencies over which the received Doppler spectrum is non-zero. If the BW of the baseband signal is much greater than B_d, then the effect of Doppler spread is negligible at the receiver. However, increasing the Doppler spread relative to the signal BW increases the signal distortion.

• *Channel coherence time:* The coherence time T_C of the channel is a statistical measure of the time duration over which the channel impulse response is essentially invariant. Thus any two signals received at different times within T_c time duration have a strong amplitude correlation. The coherence time is commonly defined as the time over which the time correlation function is above 0.5 as

$$T_C \approx \frac{0.423}{f_D}$$

where f_D is the maximum Doppler frequency. Note that T_c and Doppler spread are inversely related. If $T_S > T_C$, then the channel conditions may change significantly during the transmission of the signal, thus the transmitted signal is likely affected differently by the channel causing distortion at the receiver. As long as the symbol rate is greater than

$1/T_C$, the channel will not cause distortion due to the relative motion of the transmitter and receiver. Distortion could also result from multipath time delay spread, depending on the channel impulse response.

• *Fast and slow fading:* The time-varying behaviour classifies a channel into fast fading or slow fading. When an RF pulse is transmitted while the mobile unit (MU) is moving, the motion of the MU increases or decreases all frequency component by up to f_D Hz. If $W \gg f_D$, the channel characteristics will change very slowly. In this case the Doppler spread of the channel is much less than the bandwidth of the baseband signal, and the signal undergoes slow fading. On the other hand, for transmission at a very slow data rate (i.e., the pulse duration is large), a MU observe fast fading if the signal bandwidth is less than the maximum Doppler frequency shift. Thus, the velocity of the mobile together with the baseband signaling determines whether a signal undergoes fast fading or slow fading. Similarly, if $T_c \gg T_s$, the time duration over which the channel remains correlated is long compared to the symbol period, and the channel is said to produce *slow fading*. In this case, the channel can be regarded as quasistatic over T_C since the channel impulse response changes at a rate much slower than the rate of change of the transmitted signal. If $T_c < T_s$, the channel is called *fast fading* causing severe distortion.

4.2 Channel Models

Let $s(t)$ be the equivalent low-pass signal transmitted over the channel and let $S(f)$ denote its frequency content. Then the equivalent low-pass received signal, exclusive of additive noise, is

$$y(t) = \int_{-\infty}^{\infty} c(t;\tau)s(t-\tau)\mathrm{d}\tau = \int_{-\infty}^{\infty} C(t;f)S(f)\, e^{j2\pi ft}\mathrm{d}f. \tag{4.1}$$

If the bandwidth W of $S(f)$ is much smaller than the coherence bandwidth of the channel (i.e., $W \ll B_c$), since $B_c \propto 1/T_m$ then $W \ll 1/T_m$ (or equivalently $T_s \gg T_m$). Hence, the delay associated with the ℓ-th multipath component $\tau_\ell \leq T_m$ and the multipath components of the channel are not *resolvable*. Thus all the frequency components of transmitted signal $S(f)$ are affected by the channel in approximately the same way (i.e., undergo the same attenuation and phase shift in transmission through the channel) and there is little time spreading in the received signal (i.e., $s(t-\tau_i) \simeq s(t)$). This implies that, within the bandwidth W occupied by $S(f)$, the time-variant transfer function $C(t;f)$ of the channel is constant in the frequency variable. Thus the frequency independent channel

response can be written as $C(t;f) = C(t)$. Such a channel that has a constant gain and a linear phase response over a bandwidth larger than the bandwidth of the transmitted signal is called *frequency-nonselective* or flat fading. Flat-fading channels are also called *narrowband channels* because the signal BW is narrower than the (coherence) channel BW. In this case the spectral characteristics of the transmitted signal remain intact at the receiver. For the frequency-nonselective channel $y(t)$ can be written as

$$y(t) = C(t)\int_{-\infty}^{\infty} S(f)e^{j2\pi ft}\mathrm{d}f = C(t)s(t) = a(t)e^{j\phi(t)}s(t) \tag{4.2}$$

where $a(t)$ represents the complex envelope and $\phi(t)$ represents the phase of the equivalent low-pass channel response. Equation (4.2) shows that the received signal is simply the transmitted signal multiplied by an appropriate stochastic process, which represents the time-variant characteristics of the channel. Thus a frequency-nonselective fading channel has a time-varying *multiplicative* effect on the transmitted signal.

When the transmitted signal bandwidth approaches or surpasses the coherence bandwidth of the mobile channel (i.e., $W > B_c$), then the frequency components of $S(f)$ with frequency separation exceeding B_c are subjected to different gains and phase shifts. The received signal includes multiple versions of the transmitted waveforms, attenuated (depending on the phases of the received overlapping signals, the signal copies may amplify or attenuate each other) and delayed in time (resulting in pulse broadening), and hence the received signal is distorted (i.e., the signal interferes with itself). Also, as each transmitted symbol is received several times, each received symbol will be distorted by adjacent symbols in the sequence causing ISI. Equivalently, when the channel impulse response has a delay spread greater than the symbol period of the transmitted signal (i.e., $T_m \gg T_s$), the multipath components extend beyond the symbol duration. In this case, the transmitted signal reaches the receiver via $L \geq 1$ directions where the relative arrival delay of at least two multipath components is greater than the period of the transmitted signal and thus two rays are resolvable (time-differentiable). This leads to the *time dispersion* of the transmitted symbols within the channel and, hence, the channel amplitude varies widely across the signal bandwidth, resulting in ISI. In fact, the channel amplitude values at frequencies separated by more than the coherence bandwidth are roughly independent. Thus, certain frequency components in the received signal spectrum have greater gains than others. The channels exhibit *frequency-selective* fading, which is also called *wideband channels* since the signal BW is wider than the BW of the channel. The dividing line between frequency-

selective and flat fading is not perfectly sharp. For example, at very low data rates, the pulse duration is high and the channel primarily slow and flat. If the data rate is very high and the MU is moving slowly, the channel will be slow but frequency-selective. If the data rate is high and the MU is moving at very high speed, the channel will be fast and frequency-selective.

The time-varying channel impulse response and the corresponding time-variant transfer function can be written as

$$\begin{aligned} c(t;\tau) &= \sum_{\ell=1}^{L} c_\ell(t)\,\delta(\tau - \ell/W) \\ C(t;f) &= \sum_{\ell=1}^{L} c_\ell(t)\,e^{j2\pi f\ell/W} \end{aligned} \tag{4.3}$$

where $c_\ell(t)$ is the complex-valued channel gain of the ℓ-th multipath component and L is the number of resolvable multipath components. Note that in order for two paths to be time-differentiable (resolvable), their relative arrival delay must be greater than the inverse of the bandwidth of the transmitted signal. Since the transmitted signal has a bandwidth of W, the delay resolution of the measurement is approximately $1/W$. Hence, only the multipath components in the channel response that are separated in delay by at least $1/W$ are resolvable. The complex signals that combine at the receiver within less than $1/W$ time period are not individually resolvable because the receiver cannot resolve delay differences smaller than $1/W$. The unresolved multipath components may be considered as clusters on the delay axis.

To model the effects of L multipath fading components on a transmitted signal $x(t)$ over short propagation distances (delays), the signal needs to be convolved with L (uncorrelated) complex-valued channel gain (attenuation) coefficients $\{c_\ell(t)\}$, $\ell = 1, \cdots, L$ [115]. Each multipath component signal $c_\ell(t)$ can be considered to be composed of many unresolvable paths. Each individual unresolvable signal in a given resolvable path has a random associated phase (due to the different propagation distances) and $c_\ell(t)$ is usually modeled as complex Gaussian random process by the virtue of the Central Limit Theorem. The randomly time-varying tap gains $\{c_\ell(t)\}$ (also called *fading signals*) may also be represented by $\{c_\ell(t)\} = a_\ell(t)e^{j\phi_\ell(t)}$ where $\{a_\ell(t)\}$ represent the amplitudes and $\{\phi_\ell(t)\}$ represent the corresponding phases. A frequency-selective channel with the complex baseband impulse response given in (4.3) can be modeled statistically by a linear finite-duration impulse response (FIR) filter with memory length L that models the summation of multiple atten-

uated and delayed component signals at any instant of time. In this model, the channel has a constant gain and linear phase response over a bandwidth smaller than that of the transmitted signal. Each path delay corresponds to the delay spread of the channel and its value can be chosen in accordance to the delay power profile of the desired channel environment. Since the delay spread of the channel is T_m and the time resolution of the multipath is $1/W$, the maximum number of taps required by the transversal filter model is given by $L = \lfloor T_m W \rfloor + 1$. Even though the path delays can be considered as random processes, the arrival times of rays are actually not random since obstacles and buildings tend to be grouped together. The *tapped delay line* (TDL) model shown in Figure 4.1 is typically used to model a multipath time-varying channel in which the L-ray multipath fading channel model receives baseband signal $s(t)$ and produces convolved samples $y(t)$ at the output. The transmitted signal is modulated in amplitude and phase by a baseband tap-gain function $c_\ell(t)$ and L delayed and modulated signals are summed to form the output signal.

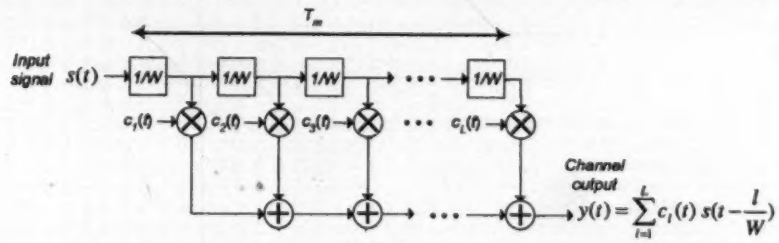

Figure 4.1: Architecture of a frequency-selective fading simulator.

In a multipath channel with L resolvable path, the channel output (the input signal to the mobile receiver) is

$$y(t) = \sum_{\ell=1}^{L} c_\ell(t) s\big[t - \tau_\ell(t)\big]$$

where $c_\ell(t)$ and $\tau_\ell(t)$ represent the time-varying attenuation and the propagation delay associated with the ℓ-th multipath component, respectively. In order to determine the complex envelope of the received signal, assume that the channel input (the transmitted signal) is a modulated signal of the form

$$s(t) = x(t)\cos\big[\omega_c t + \phi(t)\big] = \Re\{s_l(t)\, e^{j\omega_c t}\}$$

where $s_l(t) = x(t)\, e^{j\phi(t)}$ is the equivalent baseband representation of $s(t)$. Then $y(t)$ can

be written as

$$\begin{aligned} y(t) &= \sum_{\ell=1}^{L} c_\ell(t) x\big[t-\tau_\ell(t)\big] \cos\Big\{\omega_c\big[t-\tau_\ell(t)\big] + \phi\big[t-\tau_\ell(t)\big]\Big\} \\ &= \sum_{\ell=1}^{L} c_\ell(t) x\big[t-\tau_\ell(t)\big]\, \Re\Big\{e^{j\omega_c t}.e^{-j\omega_c\tau_\ell(t)}.e^{j\phi\left[t-\tau_\ell(t)\right]}\Big\}. \end{aligned}$$

Assuming the complex envelope of the transmitted signal is denoted by $x_l(t) = x(t)\, e^{j\phi(t)}$, then

$$x_l\big[t-\tau_\ell(t)\big] = x\big[t-\tau_\ell(t)\big] e^{j\phi\left[t-\tau_\ell(t)\right]}$$

so that

$$y(t) = \Re\left\{\sum_{\ell=1}^{L} c_\ell(t) x_l\big[t-\tau_\ell(t)\big] e^{-j\omega_c\tau_\ell(t)}\, e^{j\omega_c t}\right\}.$$

The complex path attenuation is defined as $\tilde{c}_\ell(t) = c_\ell(t)\, \exp\big[-j2\pi f_c\tau_\ell(t)\big]$ so that

$$y(t) = \Re\left\{\sum_{\ell=1}^{L} \tilde{c}_\ell(t)\, x_l\big[t-\tau_\ell(t)\big] e^{j\omega_c t}\right\}.$$

Thus, the complex envelope of the receiver input is

$$y_l(t) = \sum_{\ell=1}^{L} \tilde{c}_\ell(t)\, x_l\big[t-\tau_\ell(t)\big]. \tag{4.4}$$

The channel input-output relationship is generally characterized as a LTV having a particular complex baseband impulse response $\tilde{c}(t;\tau)$ measured at time t assuming that the impulse is applied at time $t-\tau$ (i.e., the path delay is τ):

$$\tilde{c}(\tau,t) = \sum_{\ell=1}^{L} \tilde{c}_\ell(t)\, \delta\big[t-\tau_\ell(t)\big].$$

In a frequency-flat channel, it is assumed that over small-scale distances and in absence of a LOS path, the scattered components arriving at the receiver will experience similar attenuations, phase shifts, and delays. Thus the $\tilde{c}_\ell(t)$'s are thus approximately equal and have a complex amplitude $a_\ell(t)e^{j\phi_\ell(t)}$. The resulting combined envelope A and phase ϕ at a single point in space is thus given by

$$\tilde{c}(t) = \sum_{\ell=1}^{L} a_\ell(t)\, e^{j\phi_\ell(t)} = A\, e^{j\phi}.$$

The time-varying nature of the channel arises from either the transmitter, or the receiver, or changes in the propagation environment. In the absence of movement or other changes

in the transmission medium, the input-output relationship is time-invariant even though multipath may be present. A time-invariant multipath channel can be represented in the time domain by an impulse response of the form

$$\tilde{c}(\tau) = \sum_{\ell=1}^{L} \tilde{c}_\ell \, \delta(\tau - \tau_\ell).$$

4.3 Stochastic Models for Fading Channels

When the propagation path involves two-dimensional (2-D) isotropic scattering with an omnidirectional receiving antenna at the receiver, the path phases can be assumed to be uniformly distributed over $(0, 2\pi)$ (i.e., the phases are randomized because of the varying path lengths) [113]. In order to obtain the distribution of the envelope sum of a large number of sinusoids with constant amplitude and uniformly-distributed random phases, a resolvable path can be considered to be composed of many unresolvable signal components. According to the CLT [8, 14], the sum of a large number of signals with constant amplitudes and uniformly-distributed random phases produces a signal that has a Gaussian distribution with a zero mean. Thus, $c(t)$ can be represented as a complex Gaussian process in t where $\Re\{c(t)\}$ and $\Im\{c(t)\}$ (the real and imaginary parts, respectively) are independent zero-mean Gaussian [133] with equal variance σ^2. Thus the envelope $|c(t)| = \sqrt{c_i(t)^2 + c_q(t)^2}$ follows the *Rayleigh distribution* $f_{|C|}(c) = \frac{c}{\sigma^2} \exp\left[-c^2/(2\sigma^2)\right]$ with a mean and variance of $\sigma\sqrt{\pi/2}$ and $\sigma^2(2 - \pi/2)$, respectively, where σ^2 is the time-averaged power of the received signal and $\phi(t) = \tan^{-1}(\frac{c_q(t)}{c_i(t)})$ is the phase of the received waveform. Since the processes $c_i(t)$ and $c_q(t)$ are Gaussian, it can be shown that $\phi(t)$ has a uniform distribution, $\phi \in (-\pi, \pi)$ [25].

While the pdf of $|c(t)|$ describes the distribution of the *instantaneous* values of the complex impulse response, other temporal and spatial variations of multipath components must be characterized for accurate fading channel modeling. Since the orientation and material properties of the obstacles between the transmitter and receiver are not in general known in advance, or may be time-varying, it is common to model the tap gains $\{c_\ell(t)\}$ as a *stochastic process* to characterize the real channels. Specifically, the tap gains are usually modeled as *wide-sense stationary* (WSS) in the t-variable and mutually uncorrelated random processes [16, 24, 25, 113]. Therefore, their statistical properties can be completely described by first-order and second-order statistics [14]. When the channel is defined as a WSS random process in t, the channel correlation function is $\mathrm{R_c}(\tau; t_1, t_2) = \mathrm{R_c}(\tau; \Delta t)$ [113, 132].

Thus, the channel ACF can be written as

$$\mathrm{R}_{c,c}(\tau_1, \tau_2, \Delta t) = \mathrm{E}\big[c^*(\tau_1, t)c(\tau_2, t + \Delta t)\big].$$

When the attenuation and phase shift associated with different delays (i.e., resolvable paths) can be assumed to be uncorrelated, the channel is said to exhibit *uncorrelated scattering* (US) in the delay τ if the channel correlation function for any two paths with delays τ_1 and τ_2 is zero when $\tau_1 \neq \tau_2$:

$$\frac{1}{2}\mathrm{E}\big[c(\tau_1; t_1)c^*(\tau_2; t_2)\big] = \mathrm{R}_c(\tau_1; t_1, t_2)\delta(\tau_2 - \tau_1).$$

where $\mathrm{E}[\cdot]$ denotes expectation. The *wide-sense stationary and uncorrelated scattering* (WSSUS) assumptions leads to

$$\mathrm{R}_{c,c}(\tau_1, \tau_2, \Delta t) = \mathrm{R}_{c,c}(\tau_1, \Delta t)\,\delta(\tau_1 - \tau_2).$$

The WSSUS model was originally proposed by Bello [113] and has been reasonably used for modeling of wireless fading channels over bandwidths up to 10 MHz [21].

A doubly-spread WSSUS channel may be characterized completely by two sets of parameters: the Doppler power spectrum (DPS) and the delay power spectrum (dPS). The DPS provides statistical information on the variation of the frequency of a pulse received by a MU. A dPS identifies the average power level of each multipath and the time delays between successive multipath components. Both parameter sets can be described by a single function, called the *delay-Doppler power spectrum* or *scattering function* $\mathcal{S}(\tau, f)$, which is a measure of the power spectrum of the channel at delay τ and frequency offset f (relative to the carrier frequency). The dPS and the DPS are defined by averaging of $\mathcal{S}(\tau, f)$ over f and τ, respectively, as follows:

$$\begin{aligned} \mathrm{dPS}(\tau) &= \mathcal{S}_c(\tau) = \int_{-\infty}^{\infty} S(\tau, f)\,\mathrm{d}f \\ \mathrm{DPS}(f) &= \mathcal{S}_c(f) = \int_{0}^{\infty} S(\tau, f)\,\mathrm{d}\tau. \end{aligned} \tag{4.5}$$

The delay spread of the channel can also be defined as the range of values over which the $\mathrm{dPS}(\tau)$ is nonzero (the width of the dPS). As a MU moves through a dispersive environment, the width and shape of the dPS can change significantly. Similarly, the Doppler spread B_d of the channel can be defined as the range of values over which $\mathrm{DPS}(f)$ is nonzero (the width of the DPS). The Doppler spread provides a measurement of the fading rate of the

channel. As shown in Figure 4.1, the delay spread of the mobile channel is implemented by delaying each multipath component by a programmable value that may be selected based on the delay power profile of the simulated channel.

The spectrum of the received signal depends on the assumptions made about the AOA statistics and the radiation pattern of the receiving antenna. M. J. Gans introduced a DPS in 1972 [25] assuming that the propagation path involves 2-D isotropic scattering with an omnidirectional antenna at the mobile receiver that signals came from all directions with uniformly distributed phases over $(0, 2\pi)$. Based on the flat fading channel model developed by R. H. Clarke in 1968 [24], the spectral density of the complex envelope of the received signal that depends on the antenna pattern is given by

$$G_{c_i}(f) = \begin{cases} \frac{1}{2\pi f_D} \frac{1}{\sqrt{1-(f/f_D)^2}} & |f| < f_D \\ 0 & |f| \geq f_D \end{cases} \tag{4.6}$$

and is known as *Jakes power spectral density* or *Jakes power spectrum*. As shown in Figure 4.2, the PSD associated with the in-phase (or quadrature) portion of the received fading signal has the well-known U-shaped bandlimited form. The PSD is centered on the carrier frequency, is zero outside the limits of $f = f_c \pm f_D$, and is infinite at f. A U-shaped power spectrum shows that most of the energy is concentrated around the f_D, however, the probability of components arriving at exactly 0° or 180° is zero due to the uniform scattering model approximation. Thus infinite PSD values are approached but never reached.

A frequency-flat channel is usually modeled as a time-correlated Gaussian WSSUS process with the complex envelope $c(t) = c_i(t) + jc_q(t)$ [113]. The temporal variation of the channel can be characterized by the ACF of $c(t)$ in the t variable. Specifically, the important properties of fading channel models are manifested in the autocorrelations, $R_{c_i,c_i}(\tau)$ and $R_{c_q,c_q}(\tau)$, and the cross-correlation $R_{c_i,c_q}(\tau)$ of the $c_i(t)$ and $c_q(t)$ components of $c(t)$, the autocorrelation $R_{c,c}(\tau)$ of the complex envelope of $c(t)$, and the autocorrelation of the squared envelope $R_{|c|^2,|c|^2}(\tau)$ [23, 24, 37]. The ACF can be obtained by taking inverse Fourier transform of the PSD given in Equation (5.10) for 2-D isotropic scattering with an omnidirectional antenna at the mobile receiver with uniformly distributed phases as follows:

$$\begin{aligned} R_{c_i,c_i}(\tau) &= R_{c_q,c_q}(\tau) = E[c_q(t)c_q(t+\tau)] = \mathcal{J}_0(2\pi f_D \tau) \\ R_{c_i,c_q}(\tau) &= R_{c_q,c_i}(\tau) = 0 \\ R_{c,c}(\tau) &= E[c(t)c^*(t+\tau)] = 2\,\mathcal{J}_0(2\pi f_D \tau) \\ R_{|c|^2,|c|^2}(\tau) &= E[|c(t)|^2|c(t+\tau)|^2] = 4 + 4\,\mathcal{J}_0^2(2\pi f_D \tau) \end{aligned} \tag{4.7}$$

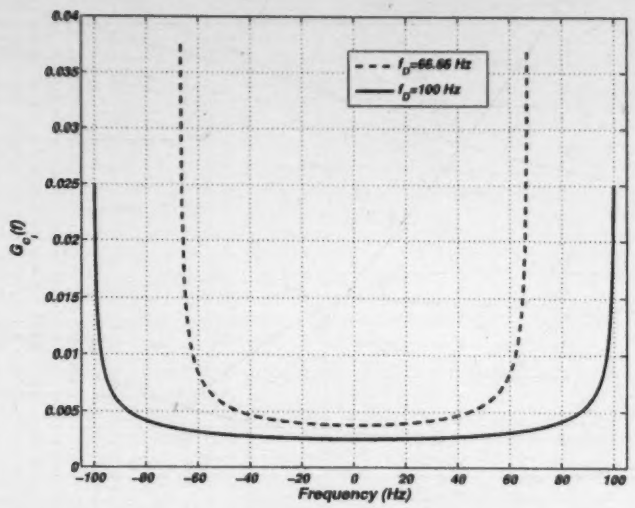

Figure 4.2: The PSD for a Rayleigh fading channel with different Doppler frequencies.

where f_D is the maximum Doppler frequency, τ is the time lag, and

$$\mathcal{J}_0(x) = \frac{1}{\pi}\int_0^{\pi} e^{-jx\cos\theta}d\theta$$

is the zero-order Bessel function of the first kind [14]. The ACF depends on such factors as the maximum Doppler frequency normalized by the sampling rate, the antenna characteristics and the propagation path [134]. According to the $\mathcal{J}_0(2\pi f_D\tau)$ plot shown in Figure 4.3, the autocorrelation is zero for $f_D\tau = 0.4$ or, equivalently, for $v\tau \approx 0.4\lambda$ where v is the velocity of MU and λ is the signal wavelength. Thus, the signal decorrelates over a distance of approximately one half of a wavelength, under the uniform path phases assumption. This approximation is commonly used as a rule of thumb to determine many system parameters of interest.

4.4 Analysis of SOS-Based Fading Channel Models

The goal of any channel simulator should be to reproduce the desired properties in (4.7). Thus to simulate multiple fading processes that are correlated in time, but uncorrelated between processes, the Rayleigh processes should fulfill the following conditions: (1) the in-phase and quadrature components of each underlying complex Gaussian random process are zero-mean independent Gaussian processes with identical variances and identical ACFs;

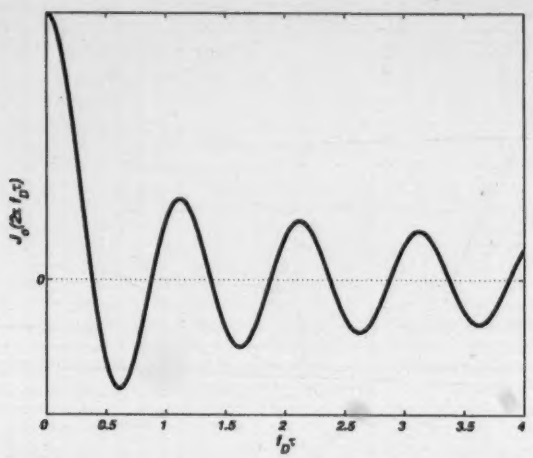

Figure 4.3: The zero-order Bessel function of the first kind.

(2) the ACFs between the $c_i(t)$ and $c_q(t)$ components are not functions of t, (i.e., the fading signal is WSS); and (3) the CCF of any pair of fading processes must be zero.

One of the approaches to approximate the fading process with the desired statistical properties in (4.7) is based on the incoherent superposition of independent complex-valued signals. This model is known as the *sum-of-sinusoids* approach and is based on *Rice theorem* [116], which implies that a Gaussian process can be modeled by the superposition of an infinite number of weighted harmonic functions with equidistant frequencies and random phases. The main idea of a SOS channel model is to simulate the fading channel as a WSSUS [113, 135] complex Gaussian random process, formed by the sum of multiple sinusoidal waveforms having amplitudes, frequencies, and phases that are appropriately selected to accurately reproduce the desired statistical properties in (4.7).

Clarke proposed a useful mathematical model for the complex channel gain, under the narrow-band flat fading assumption ($\tau_n = 0 \; \forall n$) [24]. Clarke showed that the complex channel gain $c(t)$ at a time t can be expressed as

$$c(t) = \sqrt{\frac{2}{N}} \sum_{n=1}^{N} \exp\left[j(2\pi f_D t \cos(\alpha_n) + \phi_n)\right] \tag{4.8}$$

where f_D is the maximum Doppler frequency, α_n and ϕ_n are the angle of arrival and the initial phase, respectively, associated with the n-th sinusoid, N is the total number of sinusoids [24], and each sinusoid has equal average amplitude (the same received power).

The phase angles ϕ_n and α_n are assumed to be mutually independent and uniformly distributed over $(-\pi, \pi)$ for all n. For sufficiently large N, according to CLT the real part $c_i(t) = \Re\{c(t)\}$ and the imaginary part $c_q(t) = \Im\{c(t)\}$ of the complex envelope are zero-mean Gaussian and independent (thus the envelope $|c(t)|$ is Rayleigh distributed). The squared envelope correlation of $c(t)$ in (4.8) can be written as [8]

$$\mathrm{R}_{|c|^2|c|^2}(\tau) = \mathrm{E}[|c(t)|^2\,|c(t+\tau)|^2] = 4 + 4\frac{N-1}{N}\mathcal{J}_o^2(2\pi f_D \tau) \tag{4.9}$$

where when N approaches infinity, the squared envelope correlation asymptotically reaches the desired value $4 + 4\mathcal{J}_o^2(2\pi f_D \tau)$. Due to the accurate statistical properties of Clarke's model, it has been widely used for Rayleigh fading channels and is sometimes referred to as the mathematical reference model.

Numerous sum-of-sinusoids models have been proposed [16, 28, 29, 31–39] based on Clarke's model. These models can be broadly categorized as either *deterministic* or *statistical* [8]. In *deterministic* SOS simulators [16, 31–34], all the waveform parameters (i.e., amplitude, Doppler frequency and phase) are known and established only once before the simulation starts and are held constant for all subsequent simulation trials. Hence, the properties of the generated signal are deterministic. On the other hand, in the *statistical* models (also called *Monte Carlo SOS* models) at least one of the waveform parameters is taken to be a random variable that changes for every simulation trial, and so the statistical properties of the generated signal change for each simulation trial, but converge statistically to the desired properties over a large number of simulation trials [28]. Since these SoS methods converge statistically to the desired properties, it is important to determine the number of simulation trials needed to achieve a desired convergence level. This is directly related to the variation in the time-average properties of a single simulation trial from the desired ensemble average properties.

Jakes proposed his deterministic SOS-based model based on Clarke's model [16] to generate time-correlated Rayleigh fading variates. The Jakes model is more computationally efficient than Clarke's model in which the in-phase $c_i(t)$ and quadrature $c_q(t)$ components of a stationary complex Gaussian process $c(t)$ are formed by a *finite* superposition of sinusoids having frequencies and phases that are appropriately chosen to accurately produce a sequence of correlated fading variates with the desired statistical properties. The model assumes that N waveforms with equal power arrive at the moving receiver with uniformly distributed arrival angles $\alpha_n = 2\pi n/N$, such that waveform n experiences a Doppler shift

f_{D_n}. Note that each Doppler frequency shift f_{D_n} has four phase shifts associated with it except for the maximum Doppler shift, which has only two phase shifts. For example, as shown in Figure 4.4, sinusoids 1, 4, 6, and 9 experience the same Doppler shift, rays 1 and 9 a positive Doppler shift, and rays 4 and 9 a negative Doppler shift. Similar conditions hold for the sinusoids 2, 3, 7, and 8. Sinusoid 5 has the maximum positive Doppler shift and ray 5 has the minimum negative Doppler shift. Thus, there is a four-fold symmetry in the magnitude of the Doppler shift, except for $\alpha = 0$ and $\alpha = \pi$.

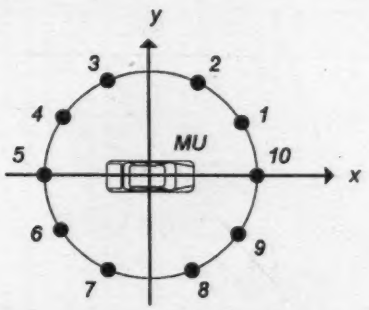

Figure 4.4: Symmetry of receiving sinusoids in Jake's design.

Despite the extensive acceptance and application of Jakes simulator, for simulation results to be meaningful, they must reproduce the important statistics of the real world. It was recently shown that the assumptions and simplifications made by Jakes adversely affect the statistics of the SOS-based fading channel simulator [29, 32, 34]. One problem with Jakes' method is that the cross-correlation between the in-phase and quadrature components are significantly different from zero [34]. Also, when the number of sinusoids is finite, the ACF is accurate only up to $[0, N/(2f_D)]$. Another important problem with the Jakes model is that its output sequence averaged across the ensemble of fading channels is not WSS [34]. In [34] the Jakes model was improved by introducing *random phase shifts* in the low frequency oscillators. An intuitive justification for using this method is the fact that for small values of time t, the values produced by the low-frequency oscillators are highly correlated (they are equal at $t = 0$). By adding the random phases, this source of correlation is eliminated. The WSS Jakes model proposed in [34] has the desired complex envelope autocorrelation as the number of sinusoids approaches infinity. However, consistent with Pop and Beaulieu's caution about some other second-order statistical properties of their proposed model [34], it was proved in [136] that the autocorrelations and cross-correlations of the quadrature components and the autocorrelation of the squared envelope

do *not* approach the desired statistics, even as $N \to \infty$. Pätzold also proposed several deterministic SOS-based modeling schemes that can be applied to simulate multiple Rayleigh fading processes that are correlated in time, but uncorrelated between processes [137]. For example, the method of exact Doppler spread [33] uses a finite number of sinusoids and deterministic discrete Doppler frequencies f_{D_n}s. In order to ensure that the different processes are uncorrelated, this model defines f_{D_n}s in such a way that they are disjoint (i.e., mutually exclusive) for different processes.

Careful studies of the theoretical models are important as some models cannot be an accurate candidate for simulating the fading channels. Recently, Patel *et al.* [8] showed different inaccuracies with the well-known SOS-based models. For example, the model in [38] has non-stationary and non-Gaussian properties. Also, the squared envelope autocorrelation in [38] and [37] is derived incorrectly.

Zheng and Xiao [36] introduced randomness to the Doppler frequency, the initial phase of the sinusoids, *and* the angles of arrival to have MC simulators with desired statistical properties. The resulting complex-valued Rayleigh fading process $c(t)$ is given by

Model I:

$$\begin{aligned} c(t) &= \sqrt{\frac{2}{N}}\left\{\sum_{n=1}^{N}\cos\left[2\pi f_D t\cos\left(\frac{2\pi n-\pi+\theta}{4N}\right)+\phi_n\right]\right. \\ &\quad \left.+j\sum_{n=1}^{N}\cos\left[2\pi f_D t\sin\left(\frac{2\pi n-\pi+\theta}{4N}\right)+\varphi_n\right]\right\} \end{aligned} \tag{4.10}$$

where θ is a random variable uniformly distributed over $[-\pi,\pi)$, and ϕ_n and φ_n are statistically independent and uniformly distributed over $[-\pi,\pi)$ for $1 \le n \le N$. *Model I* has several advantages over previous simulation models such as [8]:

1. It avoids the stationarity problem while maintaining the accuracy of the correlation statistics. The ACFs of the in-phase and quadrature components and the ACFs of the complex envelope match those of Clarke's reference model very closely, even for small N. Also, the ACF of the squared envelope of the fading signal $c(t)$ is $\mathrm{R}_{|c|^2|c|^2}(\tau) = 4 + 4\mathcal{J}_o^2(2\pi f_D\tau) + \frac{2+\mathcal{J}_o(2\pi f_D\tau)}{N}$ [36]. It asymptotically approaches the desired autocorrelation as $N \to \infty$, while good approximation has been observed when N is not less than eight. The model always produces uncorrelated in-phase and quadrature components, as required for a Rayleigh-distributed envelope.

2. The autocorrelation and cross-correlation functions do not depend on N. This high-

lights the advantages of the new simulation model over all other existing simulation models.

3. When N is as small as eight, the envelope $|c|$ is Rayleigh-distributed and the phase $\Phi(t) = \arctan[c_i, c_q]$ is uniformly distributed on $[-\pi, \pi)$.

4. Due to the proper selection of the simulation parameters in the model in [36], the variance of correlation functions $\mathrm{Var}[\mathrm{R}(\cdot)] = \mathrm{E}[|\tilde{\mathrm{R}}(\cdot) - \lim_{N\to\infty} \mathrm{R}(\cdot)|^2]$ of this mode are lower than the variances for most other models for finite N [37] .

Since the improved SOS-based channel models require that a relatively small number of sinusoids ($8 \leq N \leq 12$) converge statistically to the desired properties, they are good candidates for an efficient and compact hardware implementation. For example, the channel simulator in [119] uses $N = 8$, the design in [117] uses $N = 9$, and the design in [131] uses $N = 16$. The commercially available Ascom SIMSTAR fading channel simulator uses $N = 22$ [138]. Figure 4.5 shows the ACF of the in-phase component of the model in [36] for three different small numbers of sinusoids. The figure also shows the cross-correlation of c_i and c_q for $N = 8$. It can be verified that the discrete-time approximate ACF using a larger number of sinusoids matches more closely to the ideal autocorrelation sequence $\mathrm{R}[m] = \mathcal{J}_0(2\pi f_D T_s |m|)$ while the CCF is almost zero.

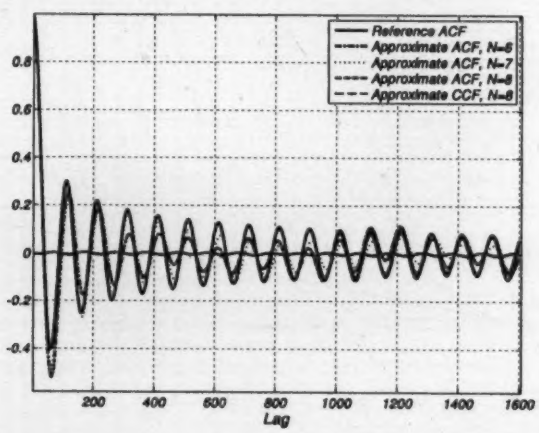

Figure 4.5: The ACF and CCF are calculated by averaging over 10 frames of 10^5 fading samples each with $f_D T_s = 0.01$.

One of the important consequences of Doppler shift is that the signal will experience deep fades. While Rayleigh statistics only provide information on the overall percentage of time that the signal goes below a certain level, it does not show how often deep fades

will actually occur. The rate at which deep fades occur is important for communication system designers as they can relate the signal level to rate of change of the received signal and velocity of the MU and choose an appropriate scheme for required data rates, designing the error control codes and diversity schemes to mitigate deep fading effects. Deep fades can be quantitatively expressed using the *level crossing rate* (LCR). The LCR is defined as the expected rate at which the magnitude (envelope) of the fading waveform crosses a threshold signal level R_{th} in the positive (or negative) going direction [115]. For the Jakes PSD, the LCR is defined as $N_R = \sqrt{2\pi} f_D \lambda e^{-\lambda^2}$ [115], where $\lambda = R_{th}/R_{rms}$ is the value of the specified threshold level R_{th}, normalized to the rms value of the fading envelope. N_R depends on f_D and thus the speed of the MU. By virtue of the factor $\lambda e^{-\lambda^2}$, there will be fewer crossing at low values of the signal level as well as at high values of the signal levels. Figure 4.6 plots the LCR of the generated fading variates for three different numbers of sinusoids. As shown in Figure 4.6, the deviation of LCR for very high Doppler rates and low crossing levels is due to the sparse sampling of the implied continuous fading waveform [139]. The envelope PDF and the CDF of 10^7 generated fading samples using

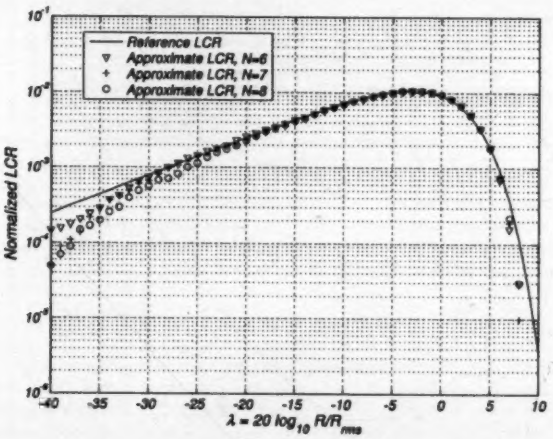

Figure 4.6: The normalized LCR calculated using 10^5 fading samples with $f_D T_s = 0.01$.

the model in (4.10) with $N = 8$ and $f_D T_s = 0.01$ is plotted in Figure 4.7.

Even though the statistical SOS-based fading simulator proposed in [36] is efficient for hardware implementations, the model is not *ergodic* and so the statistical properties of a single simulation, no matter how many samples are generated, do not converge to the reference properties. In fact, its statistical properties converge to the desired properties only when they are averaged over a large number of simulation trials and thus the channel model

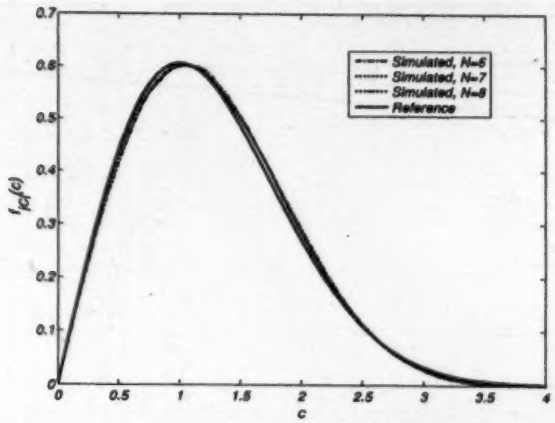

Figure 4.7: Envelope PDF for Zheng and Xiao's SOS fading channel model.

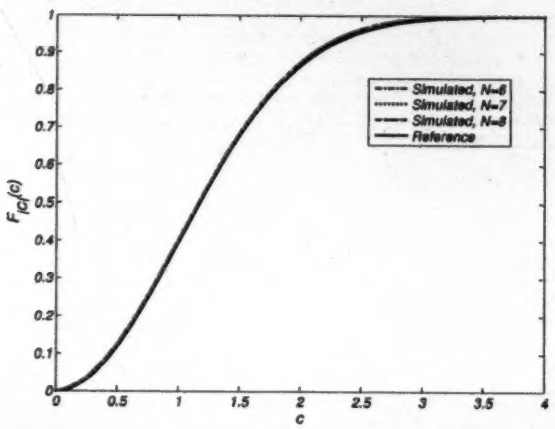

Figure 4.8: Envelope CDF for Zheng and Xiao's SOS fading channel model.

is not ergodic. If the channel model is ergodic, then the statistics of the output may converge to the reference ones in a single simulation trial. Figure 4.9 plots the ACF of the channel generated using one simulation trial with the model in (4.10) for one block containing 10^7 samples. Clearly, the ACF deviates from the reference ACF of the Rayleigh fading channel, especially at the larger lags, while the CCF stays close to zero. Therefore, *Model I* may not be suitable for simulating an ergodic Rayleigh fading channel. In Section 4.6, we propose modifications to *Model I* to overcome this limitation.

Even though model in (4.10) may not be suitable for emulating an ergodic Rayleigh fading channel, it can still be used for simulating block-based transmission systems. As shown in Figures 4.5-4.7, the statistics of the generated fading variates closely match the

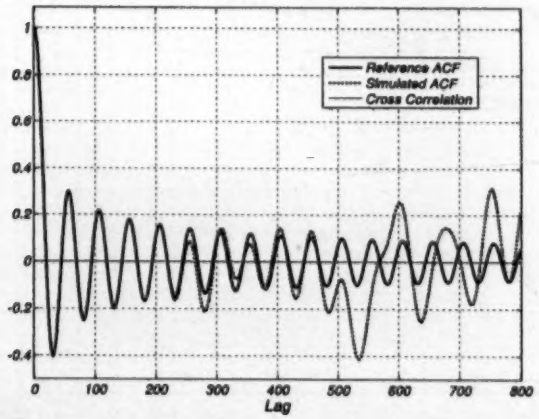

Figure 4.9: ACF and CCF for one block containing 10^7 fading samples using the Jakes SOS model from equation (4.10), $f_D T_s = 0.02$, $N = 8$.

reference curves for block-oriented fading channel models. Moreover, in some applications it is sufficient to match the ACF in a certain range only. For example, for a differential phase-shift keying (DPSK) signal transmitted over a flat-fading channel and detected by a conventional demodulator, the tail of the ACF is not relevant and the ACF must only be accurate up to T_s [140].

4.5 Implementation of an SOS Fading Channel Simulator

All of the SOS-based approaches approximate a coloured Gaussian process by a sum of low frequency oscillators. However, these models differ from one another in terms of the model parameters, which leads to differing statistical properties. Recently, Patel, Stüber, and Pratt [8] have shown that the variance in the autocorrelation of the in-phase (or quadrature) component of the complex envelope $c(t)$ produced by the SOS model in [36] is lower than the variance for most other models. The detailed comparison study in [8] concluded that the model in [36] has superior properties among all models for finite N. This fact was discussed and verified in Section 4.4. In this section a compact implementation of an SOS-based channel model is proposed that fits on a small fraction of a commonly used FPGA.

The continuous-time equations in *Model I* [36] can be written in discrete-time as follows

to simplify the computation for hardware implementation:

$$\begin{aligned} c[m] &= \sqrt{\frac{2}{N}}(c_i[m] + jc_q[m]),\ m = 1, \cdots, M \\ c_i[m] &= \sum_{n=1}^{N} \cos(2\pi(\psi_{in} m + \phi_n)) \\ c_q[m] &= \sum_{n=1}^{N} \cos(2\pi(\psi_{qn} m + \varphi_n)) \\ \psi_{in} &= f_D T_s \cos\alpha_n,\ \psi_{qn} = f_D T_s \sin\alpha_n \\ \alpha_n &= \frac{2\pi n - \pi + \theta}{4N} \end{aligned} \tag{4.11}$$

where m is the discrete time index and M is the block length. Random variables ϕ_n and φ_n, for $n = 1, 2, \cdots, N$, lie within (-0.5,0.5) and can be generated at the beginning of each fading block using two on-chip PNGs or read from an external source. $\{\psi_{in}\}$ and $\{\psi_{qn}\}$ are arrays of constant values within a fading block that can be re-initialized at the beginning of each fading block. Their value depend on a uniformly generated random variate $\theta \in (-\pi, \pi)$ and the discrete maximum Doppler frequency f_D. Note that the quality of the uniform random number generator becomes crucial specially when the number of sinusoids is small. If the randomly generated parameter set are not uniformly distributed, then the statistics become poor and characterization measurement will be incorrect.

From Equations (4.11) it is clear that an efficient and accurate implementation of the cosine function will directly improve the performance and overall accuracy of the simulator. The channel simulator in [119] uses linear interpolation to implement the sine function, while the design in [131] uses a quarter period, partitioned into 256 segments, to approximate the trigonometric functions. Fixed-point analysis provides insight into the optimization of the dataflow, and, correspondingly, the architecture of the hardware. For example, as shown in Figure 4.10 (a), since the value of $\cos(2\pi\beta) = \cos(2\pi\, 0.\beta_f)$, where β is a floating-point number in $\beta_i.\beta_f$ format, the values of $\psi_{in} m$ (and $\psi_{qn} m$) can be obtained using an adder instead of a multiplier. The adders accumulate successive values of $0.\beta_f$ for the real and imaginary components. Even for the relatively small value of N, this provides significant resource reduction in the implementation of the channel simulator. To quickly add the N quadrature components, we took advantage of the fast adder blocks now widely available in FPGAs and organized them into a pipelined tree-structured. The datapath shown in Figure 4.10(a) adds two in-phase components of $c_i[m]$ given in (4.11). This datapath also requires one dual-port ROM to simultaneously produce two values for $\cos(2\pi(\psi_{in} m + \phi_n)$

and $\cos(2\pi(\psi_{i(n+1)}m + \phi_{(n+1)})$. Figure 4.10(b) shows the tree-structured datapath for summing $N = 8$ in-phase components of $c_i[m]$. The blocks labeled *"Add two oscs"* denote instances of the circuit in Figure 4.10(a).

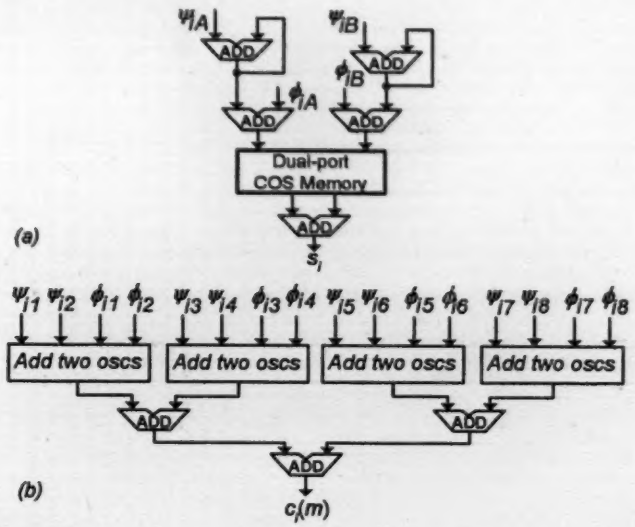

Figure 4.10: (a) Circuit for summing $N = 2$ complex oscillators. (b) Tree-structured adder for summing $N = 8$ oscillators.

To ensure computation accuracy, the fixed-point format of various signals was chosen based on experiments with different precisions that determined their impact on the statistical properties of generated fading variates. Then the HDL model of the proposed datapath was simulated to verify the accuracy of the results against the fixed-point software simulation results. For example, Figure 4.11 shows that the envelope PDF of 10^7 generated fading variates using the SOS-based model in (4.11) with 16-bit precision closely matches the theoretical curve.

To evaluate the accuracy of the fading channel simulator, the envelope statistics of the resulting sequence of complex gains were compared with those of the theoretical Rayleigh fading process. We assumed a normalized Doppler rate $f_D T_s = 0.01$, but the parameterizable model can use any arbitrary Doppler rate with 16-bit precision. Figure 4.12 plots the reference and calculated ACF and also the CCF of the generated faded envelopes for different precisions of the channel simulator datapath. The ACF and CCF plots were constructed by averaging over 10 blocks containing 10^5 sampled fading variates. The autocorrelation of the fading process generated with the 10-bit datapath shows close agreement with the

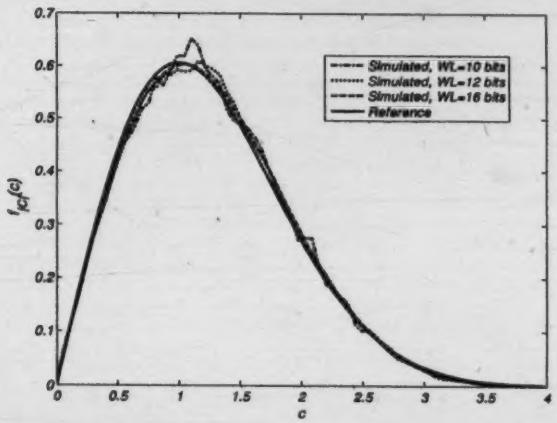

Figure 4.11: The PDF of the simulated fading envelope and their references with normalized Doppler rate $f_D T_s = 0.01$.

double-precision theoretical Bessel function, as required by Equations (4.7). Also, cross-correlation of the generated quadrature components of the complex channel gains confirms that they are relatively uncorrelated as required. Figure 4.13 plots the LCR of the fading signal generated using the fading simulator, for different precisions, against the theoretical reference. This plot verifies that the 10-bit precision is sufficient to obtain an LCR that closely matches that of the reference model. Figure 4.14 plots the CDF of the simulated fading envelope with different precisions.

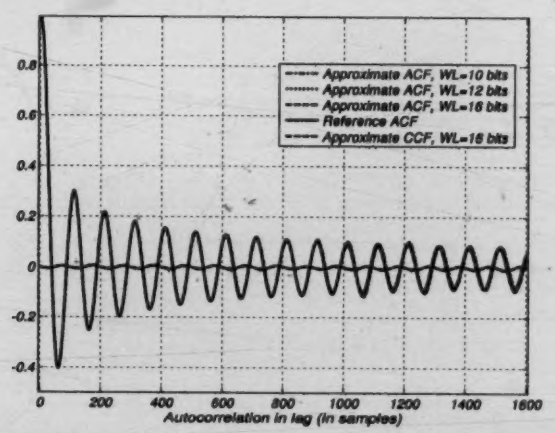

Figure 4.12: ACF and CCF of the quadrature component.

Table 4.1 summarizes the implementation characteristics for the new SOS-based fading channel simulator, with $N = 8$, on four different FPGAs. The synthesis results verify that

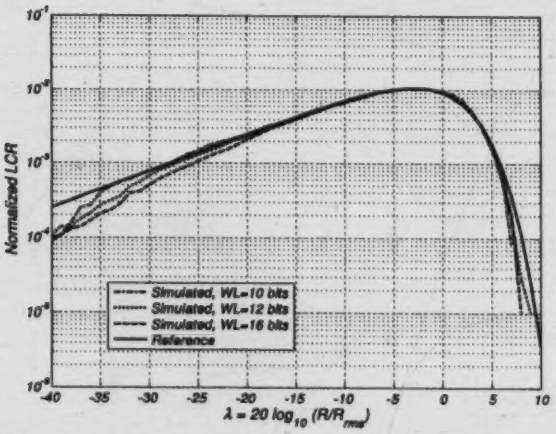

Figure 4.13: The normalized LCR of the generated fading samples.

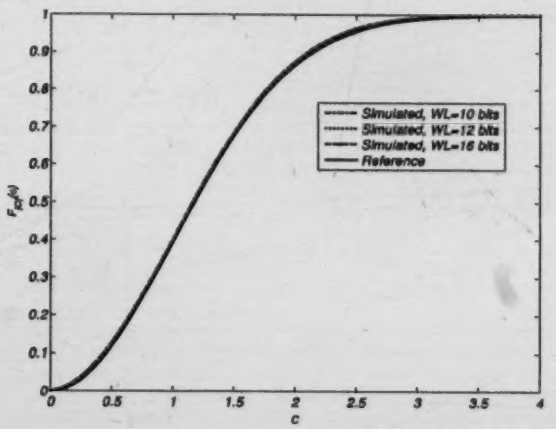

Figure 4.14: The CDF of the simulated fading envelope and their references.

the optimized datapath on the FPGA can be used to generate over 200 million complex Rayleigh fading variates per second, which is over 500 times faster than a software-based simulator written in C running on a 3.4-GHz Pentium 4 processor. Figure 4.15 shows the layout of a $356,409\,\mu m^2$ semicustom integrated circuit implementation of fading channel model designed in a 90-nm CMOS technology using a dual-threshold standard cell library. The core was targeted to operate at 500 MHz, generating 500 million complex fading variables per second while dissipating 36.9 mW of dynamic power. Static power dissipation is estimated to be 19.94 mW.

As shown in Figure 4.15, the core area is dominated by the dual port cosine ROMs. As an alternative to the table look-up method of approximating the cosine function, we propose

Table 4.1: Implementation of the fading channel simulator on different FPGAs.

Device	I[a]	II	III	IV
Clock freq. (MHz)	221.92	201.69	179.211	145.33
Output rate (MSamps/sec)	221	201	179	145
Number of slices	542	542	542	1,160
Resource utilization	0.86%	1%	2%	1%
On-chip memory blocks	8	8	8	8

[a]Design I was synthesized for a Xilinx Virtex4 XC4VFX140-11 FPGA. Design II was synthesized for a Xilinx Virtex2P XC2VP100-6 FPGA. Design III was synthesized for a Xilinx Virtex-II XC2V4000-6 FPGA. Design IV was synthesized for an Altera Stratix EP1S80F1508C6 FPGA. The latency of the fading simulator is four clock cycles in all cases.

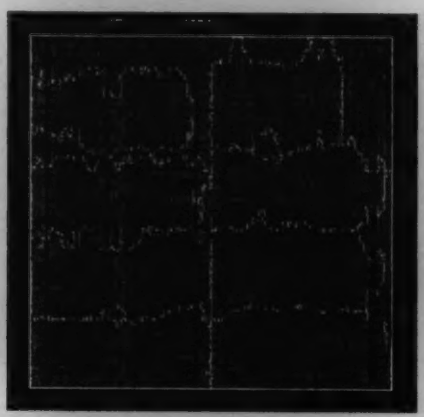

Figure 4.15: Layout of the 500 MHz semicustom fading channel variate generator.

an iterative method to calculate the in-phase and quadrature components of $c(t)$ in (4.10). The continuous-time equations in (4.10) can be written in discrete-time as follows:

$$c[m] = \sqrt{\frac{2}{N}}\Big(\sum_{n=1}^{N} I_{m,n} + j\sum_{n=1}^{N} Q_{m,n}\Big),\ m = 1, \cdots, M \qquad (4.12)$$

where $I_{m,n} = \cos(2\pi m\, f_D T_s \cos(\alpha_n) + \phi_n)$ and $Q_{m,n} = \cos(2\pi m\, f_D T_s \sin(\alpha_n) + \varphi_n)$. The n-th in-phase sinusoid can be expanded as follows:

$$\begin{aligned} I_{m,n} &= \cos(2\pi m\, f_D T_s \cos(\alpha_n) + \phi_n)) \\ &= \cos(m\, p_n + \phi_n) \ \text{ where } \ p_n = 2\pi f_D T_s \cos(\alpha_n) \\ &= \cos((m-1)\, p_n + \phi_n + p_n) \\ &= \cos((m-1)\, p_n + \phi_n)\cos(p_n) - \\ &\quad \sin((m-1)\, p_n + \phi_n)\sin(p_n) \\ &= I_{m-1,n}\cos(p_n) - C_{m-1,n}\sin(p_n) \end{aligned} \qquad (4.13)$$

$$\begin{aligned} \text{where } C_{m,n} &= \sin(2\pi m\, f_D T_s \cos(\alpha_n) + \phi_n) \\ &= C_{m-1,n}\cos(p_n) + I_{m-1,n}\sin(p_n) \end{aligned} \tag{4.14}$$

The n-th quadrature component can be expanded similarly and expressed as:

$$Q_{m,n} = Q_{m-1}\cos(p_n) - D_{m-1}\sin(p_n) \tag{4.15}$$

where $D_{m,n} = D_{m-1,n}\cos(p_n) + Q_{m-1,n}\sin(p_n)$. As given by Equations (4.13) and (4.15), the iterative calculation of $I_{m,n}$ and $Q_{m,n}$ requires the previous values $I_{m-1,n}$ and $Q_{m-1,n}$, respectively. A datapath for generating the n-th in-phase oscillator, $I_{m,n}$, is shown in Figure 4.16. Note that N instances of this datapath can be used in parallel to generate the N oscillators for the in-phase component of $c[m]$. It is important to note that for $n = 1 \cdots, N$, the values of $I_{1,n}, C_{1,n}, Q_{1,n}$ and $D_{1,n}$ must be initialized; as well, as the maximum Doppler frequency of every low-frequency oscillator must be initialized.

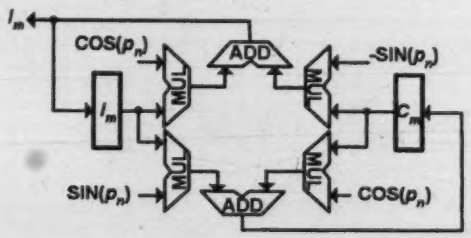

Figure 4.16: Datapath for generating one low-frequency oscillator.

Figures 4.17 and 4.18 show the ACF and CCF of the in-phase component of the iterative model for different precisions when computing $I_{m+1,n}$ and $Q_{m+1,n}$. It is evident that 10-bit precision is not acceptable while 16-bit precision provides much improved accuracy. Figure 4.19 plots the envelope PDF of 10^7 generated fading variates with three different word length precisions used to compute $I_{m,n}$ and $Q_{m,n}$. It is evident that choosing $WL \geq 28$ bits provides a close match to the reference statistics. Compared to the 16-bit precision required in the previous implementation scheme, the higher precision is required in this model to offset the accumulation of quantization error when evaluating iterative Equations (4.13) and (4.15).

Due to the iterative structure of the $I_{m,n}$ computation, the right hand side of Equation (4.13) must be computed before the next iteration. For correct operation, the multiplication and addition should be performed in one clock cycle. The shortest clock period for this datapath is limited by $t_m + t_a$, where t_m is the delay of the critical path through the

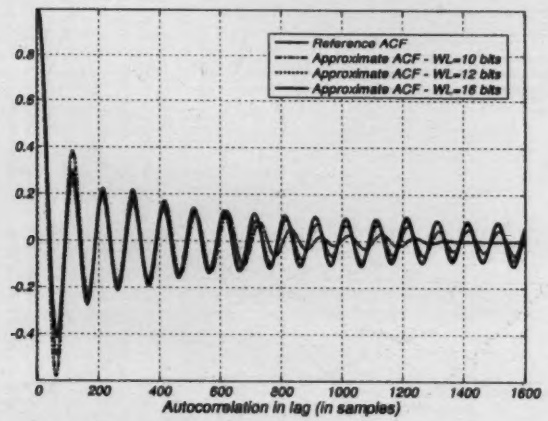

Figure 4.17: The ACF calculated by averaging over 10 frames of 10^5 fading samples with $f_D T_s = 0.01$.

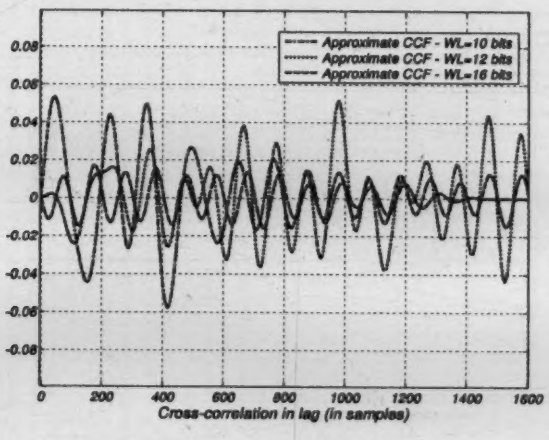

Figure 4.18: The CCF calculated by averaging over 10 frames of 10^5 fading samples with $f_D T_s = 0.01$.

multiplier and t_a is the delay of critical path through the adder. If the implementation on the Xilinx Virtex2P XC2VP100-6 FPGA uses a 28-bit datapath, then $t_m + t_a$ is about 11.4 ns and thus the channel simulators is capable of generating 87 million fading variates per second. To speed up the computation, we could use a k-stage pipelined datapath. Even though a k-stage pipelined datapath would operate at a higher clock frequency than a non-pipelined circuit, due to the iterative behaviour of the computation, the delay introduced by k pipeline registers reduces the fading variate generation rate by a factor k. For example, a four-stage pipelined version of the datapath in Figure 4.16 operates at 227 MHz, and the throughput is 227/4= 56 million fading variates per second. To avoid throughput reduction

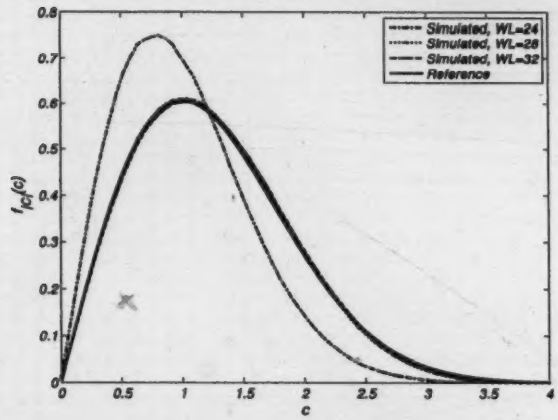

Figure 4.19: Envelope pdf of 10^7 generated fading variates with $f_D T_s = 0.01$ for three different precisions.

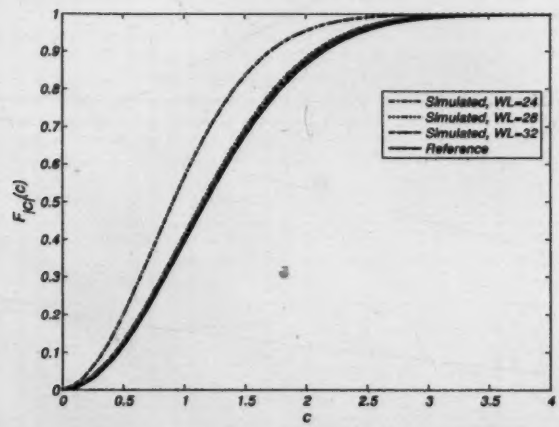

Figure 4.20: The cdf of 10^7 generated fading variates with two different precisions.

with the pipelined datapath, we can combine more than one iteration into a single stage. Using Equations (4.13) and (4.14), $I_{1,n}$ and $C_{1,n}$ can be re-written as:

$$\begin{aligned} I_{1,n} &= I_{0,n}\cos(p_n) - C_{0,n}\sin(p_n) \\ C_{1,n} &= C_{0,n}\cos(p_n) + I_{0,n}\sin(p_n) \end{aligned} \tag{4.16}$$

Similarly, $I_{2,n}$ and $C_{2,n}$ can be written as:

$$\begin{aligned} I_{2,n} &= I_{1,n}\cos(p_n) - C_{1,n}\sin(p_n) \\ C_{2,n} &= C_{1,n}\cos(p_n) + I_{1,n}\sin(p_n) \end{aligned} \tag{4.17}$$

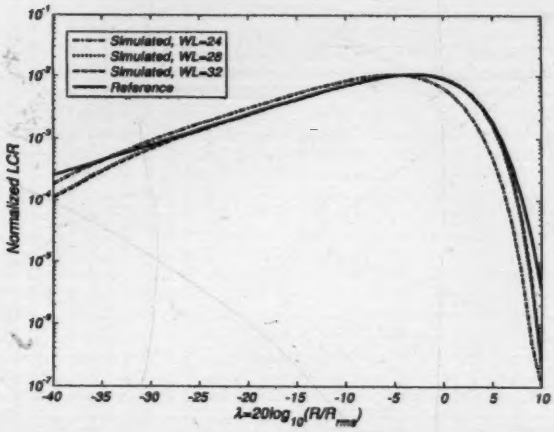

Figure 4.21: The normalized LCR of 10^7 generated fading envelopes with two different precisions.

Equations (4.16) can be substituted into Equations (4.17) to give:

$$\begin{aligned} I_{2,n} &= I_{0,n}\cos(2p_n) - C_{0,n}\sin(2p_n) \\ C_{2,n} &= C_{0,n}\cos(2p_n) + I_{0,n}\sin(2p_n) \end{aligned} \tag{4.18}$$

In general, $I_{k,n}$ and $C_{k,n}$ can be calculated as:

$$\begin{aligned} I_{k,n} &= I_{0,n}\cos(kp_n) - C_{0,n}\sin(kp_n) \\ C_{k,n} &= C_{0,n}\cos(kp_n) + I_{0,n}\sin(kp_n) \end{aligned} \tag{4.19}$$

Equations (4.19) simplify the implementation of a k-stage pipelined datapath. To avoid unused "bubble" cycles in the pipeline, the $\cos(p_n)$ and $\sin(p_n)$ constant values should be replaced with $\cos(kp_n)$ and $\sin(kp_n)$, respectively; also, the first k initial values of $I_{m,n}$ and $C_{m,n}$, for $m = 1, \cdots, k$, should be loaded into the "I_m" and "C_m" registers in k successive clock cycles, using two multiplexers and external inputs $Init_i$ and $Init_c$, respectively, as shown in Figure 4.22. The datapath shown in Figure 4.23(a) adds two of the oscillator in-phase components of $c_i[m]$ given in Equation (4.12). Figure 4.23(b) shows the tree-structured datapath for summing $N = 8$ in-phase components of $c_i[m]$. The blocks labeled "*Calc* I_0" denote separate instances of the circuit in Figure 4.22.

The complex sinusoid generator circuit was implemented in a 28-bit fixed-point format and the adder tree was implemented using 16-bit adders. Table 4.2 summarizes the implementation characteristics for the SOS-based fading channel simulator, with $N = 8$, on

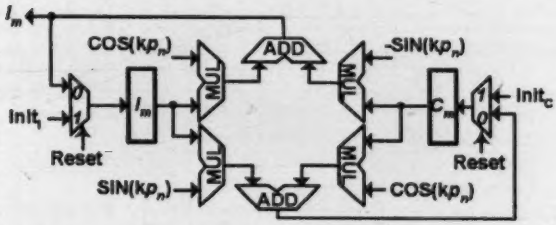

Figure 4.22: Pipelined datapath for generating one low-frequency oscillator.

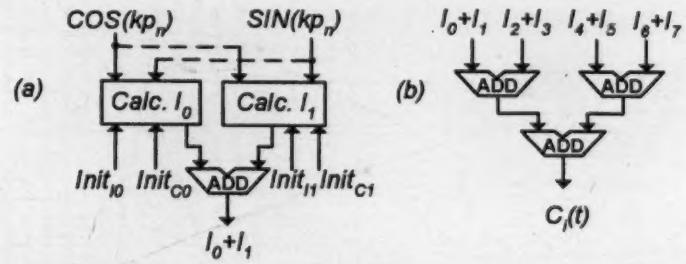

Figure 4.23: (a) Circuit for summing two complex oscillators. (b) Tree-structured adder for summing $N = 8$ oscillators.

four different FPGAs. The synthesized layout of this fading channel simulator in a 90-nm CMOS technology is 967 μm^2, when the core was targeted to operate at 500 MHz, generating 500 million complex fading variables per second. The core dissipates 185.26 mW of dynamic power and 114.4 mW of static power.

4.6 An Improved SOS-based Fading Channel Emulator

The statistical properties of a RP can be obtained through measurements. Repeating the random experiment gives rise to the random process and taking the arithmetic average of the quantities of interest. For example, to estimate the mean $m_X(t)$ of a RP $X(t, \zeta)$, we

Table 4.2: Implementation of the fading channel simulator on different FPGAs.

Device	I[a]	II	III
Clock freq. (MHz)	240.67	209.68	84.37
Output rate (MSamps/sec)	240	209	84
Number of slices	12078	12078	4,984
Resource utilization	49%	27%	30%

[a]Design I was synthesized for a Xilinx Virtex4 XC4VSX55-11 FPGA. Design II was synthesized for a Xilinx Virtex2P XC2VP100-6 FPGA. Finally, Design III was synthesized for an Altera Stratix EP1S80F1508C6 FPGA.

may repeat the random experiments N times and take the following average

$$\hat{m}_X(t) = \frac{1}{N}\sum_{k=1}^{N} X(t,\zeta_k)$$

where $X(t,\zeta_k)$ is the *realization* observed in the k-th iteration. On the other hand, the time average of a single realization can be written as

$$\tilde{m}_X(t) = \frac{1}{2T}\int_{-T}^{T} X(t,\zeta)\mathrm{d}t.$$

As the observation interval becomes large, the ergodic theorem states when the time averages converge to the ensemble average (expected value) [14]. The law of large numbers states that if $X[n]$ is an i.i.d discrete-time RP with finite mean $\mathrm{E}[X[n]] = m$, then the time average of the samples converges to the ensemble average. Thus the mean can be estimated by taking the time average of a single realization of the process. In this case $X(t)$ is *ergodic in mean*. Similarly, $X(t)$ is *ergodic in ACF* if

$$\hat{\mathrm{R}}_X(\tau) = \tilde{\mathrm{R}}_X(\tau) = \lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T} X(t)X(t+\tau)\mathrm{d}t.$$

Similarly, for a discrete-time RP $X[n]$, the time-average estimate for the mean and ACF of $X[n]$ can be expressed as

$$\begin{aligned}\tilde{m}_X[n] &= \frac{1}{2T+1}\sum_{n=-T}^{T} X[n] \\ \tilde{\mathrm{R}}_X[k] &= \frac{1}{2T+1}\sum_{n=-T}^{T} X[n+k]\,X[n].\end{aligned}$$

As discussed in Section 4.4, the channel simulator in *Model I* is a MC simulator that is WSS but not ergodic. If a model is ergodic, the statistical properties may converge to the desired ones in a single simulation trial also. If a signal is WSS and ergodic, the first and second-order time averages may be substituted for stochastic ones. However, the statistical properties of a single simulation (i.e., averaging over time) of *Model I* do not converge to the reference properties. The statistical properties of the MC simulator in *Model I* converge to the desired properties only over several simulation trials. The required number of simulation trials to achieve a desired convergence level is directly related to the variation in the time average properties of a single simulation trial from the desired ensemble average properties. The time average correlations $\tilde{\mathrm{R}}(\cdot)$ are random and depend on a specific realization of the

random parameters in a given simulation trial. The time average correlations of complex fading signal $c(t)$ can be written as

$$\begin{aligned}
\tilde{\mathrm{R}}_{c_i,c_i}(\tau) &= \lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T} c_i(t)c_i(t+\tau)\mathrm{d}t \\
\tilde{\mathrm{R}}_{c_i,c_q}(\tau) &= \tilde{\mathrm{R}}_{c_q,c_i}(\tau) = 0 \\
\tilde{\mathrm{R}}_{c,c}(\tau) &= \frac{1}{2}\lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T} c^*(t)c(t+\tau)\mathrm{d}t
\end{aligned} \tag{4.20}$$

MC SOS-based simulators are in general complex since they require a relatively large number of simulation trials for convergence. To overcome the complexity problem, a multiple parameter set MC (MPS-MC) simulation method has been proposed [140]. This technique divides a simulation trial into several frames and generates random Doppler frequencies and phases for each frame. For example, to generate 10^7 inphase and quadrature components, we can divide them into 10^3 frames of length 10^4 samples each to get time-averaged autocorrelation results. It should be noted that the autocorrelation with the MPS-MC model is zero if the time delay exceeds the frame length. Hence, the frame length should be sufficiently long to cover the time delays of interest to get meaningful results. With this method, the performance of MC models is considerably improved [8]. Unfortunately, the MPS-MC model creates discontinuities in the temporal correlation. As a consequence, the testing of a communication system should be interrupted and re-initialized every time with a new set of random parameters for each trial to ensure accurate modeling of the channel. At the receiver, the channel estimation or carrier recovery at the receiver must be re-acquired after each draw of random parameters. However stopping and restarting the communication system and channel simulator in this way might not be convenient in many practical cases.

To improve the existing MPS-MC models, Zajić and Stüber [141] proposed a deterministic model that is ergodic. However, the autocorrelations of the in-phase and quadrature components do not accurately match the theoretical properties. They also proposed a statistical model to overcome this shortcoming of their deterministic model. However, the resulting modified model is no longer ergodic.

We modified the SOS simulator proposed by Zheng, Xiao, and Beaulieu [36–38] to achieve an *improved* SOS Rayleigh fading channel simulator. In the new model, we replace the random angle of arrival by a *random walk* stochastic process [14]. Analytical analysis of our new model appears to be intractable, however, through numerical simulation we will show that the statistical properties of the new model accurately match the reference

functions. To generate fading variates with accurate statistical properties, we propose to use the following discrete *Model II*, which is a modified version of *Model I*.

Model II:

$$c[m] = c_i[m] + c_q[m]$$

$$c_i[m] = \sqrt{\frac{2}{N}} \sum_{n=1}^{N} \cos(2\pi f_{D_T} m \cos(\alpha_n[m]) + \phi_n) \tag{4.21}$$

$$c_q[m] = \sqrt{\frac{2}{N}} \sum_{n=1}^{N} \cos(2\pi f_{D_T} m \sin(\alpha_n[m]) + \varphi_n) \tag{4.22}$$

where $f_{D_T} = f_D T_s$ is the normalized maximum Doppler frequency (and T_s is the symbol period), and $\alpha_n[m] = (2\pi n - \pi + \theta[m])/(4N)$ where θ is a stationary stochastic process. Compared to *Model I*, in *Model II*, θ (and the corresponding angle of arrival) is a stochastic process rather than a random variable produced by a white process. However, great care should be taken to choose $\theta[m]$ so that the statistical properties of *Model II* match those of the reference model. In particular, through numerical experiments we found that θ must be ergodic, highly-correlated, and uniformly distributed over $[-\pi, \pi)$. Specifically, we propose to use the stochastic random walk process given by Algorithm 5. The random walk process

Algorithm 5 The proposed random walk process θ

```
Initialize δ_o = ε ≪ 1, θ[0] = U(−π, π);
for m > 0 do
    θ[m] = θ[m − 1] + δ_o × u[m];
    if θ[m] > +π then
        θ[m] = +π; δ_o = −δ_o;
    end if
    if θ[m] < −π then
        θ[m] = −π; δ_o = −δ_o;
    end if
end for
```

θ in this algorithm is generated using a white process, u, which is uniformly distributed over $[0, 1)$. The step size δ_o is chosen to be small enough to ensure that the resulting process θ is highly correlated. The step size is a function of the normalized Doppler frequency and the precision of the variables used in simulation. Numerous numerical simulations were performed to evaluate the statistical properties of *Model II*. We found different acceptable values of δ_o for various normalized Doppler frequencies as given in Table 4.3. For n-th sinusoid at time index m we can write

$$\alpha_n[m] = \frac{2\pi n - \pi + \theta[m]}{4N} = \frac{2\pi n - \pi + \theta[m-1]}{4N} + \delta' = \alpha_n[m-1] + \delta'$$

Table 4.3: Maximum step size δ_o.

Normalized Doppler frequency f_{D_T}	*Max. step size* δ_o
$f_{D_T} \leq 0.0001$	0.0000005
$f_{D_T} \leq 0.0005$	0.000001
$f_{D_T} \leq 0.001$	0.000005
$f_{D_T} \leq 0.005$	0.00001
$f_{D_T} \leq 0.01$	0.0001

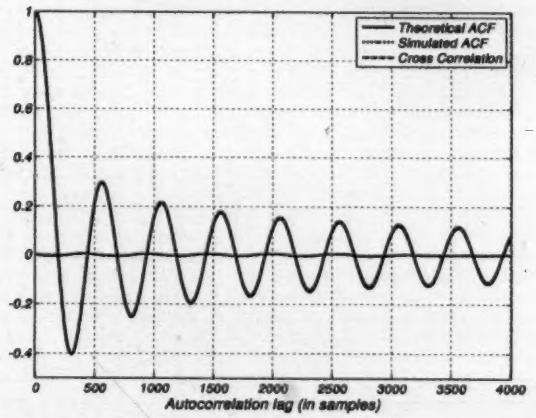

Figure 4.24: ACF and CCF of 10^7 fading variates generated by *Model II*.

and thus $\cos(\alpha_n[m])$ and $\sin(\alpha_n[m])$ can be approximated as,

$$\begin{aligned} \cos(\alpha_n[m]) &\simeq \cos(\alpha_n[m-1]) - \delta' \sin(\alpha_n[m-1]), \\ \sin(\alpha_n[m]) &\simeq \sin(\alpha_n[m-1]) + \delta' \cos(\alpha_n[m-1]). \end{aligned}$$

A block of 10^7 fading samples using $N = 8$ sinusoids with $f_D T_s = 0.002$ was generated (in one simulation trial) and the statistical properties of *Model II* were measured. Figure 4.24 plots the ideal ACF along with the ACF and CCF of the samples generated with the new model. As Figure 4.24 shows, the generated ACF accurately matches the ideal ACF and the generated CCF is very small. The LCR [23] of the envelope of the generated fading variates and the theoretical LCR are plotted in Figure 4.25. Here again a close match between the generated LCR and the desired LCR can be observed. Also, Figures 4.26 and 4.27 plot the PDF and the CDF of the generated fading variates against the reference functions. These plots show that *Model II* can faithfully reproduce the properties of Clarke's model. In the next section we focus on the hardware implementation of the fading channel emulator.

The proposed fading channel emulator was implemented as a Verilog hardware descrip-

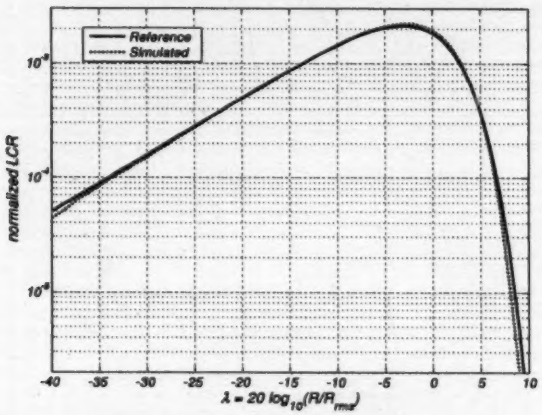

Figure 4.25: LCR of 10^7 fading variates generated by *Model II*.

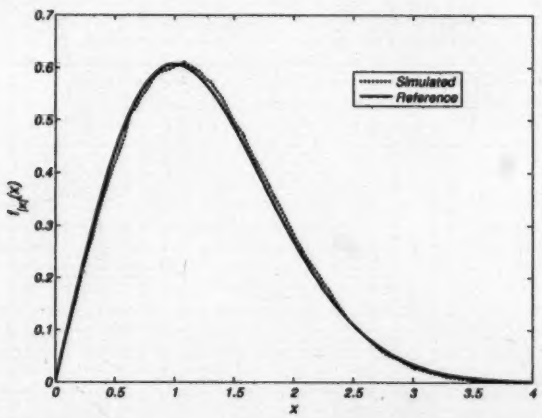

Figure 4.26: PDF of 10^7 fading variates generated by *Model II*.

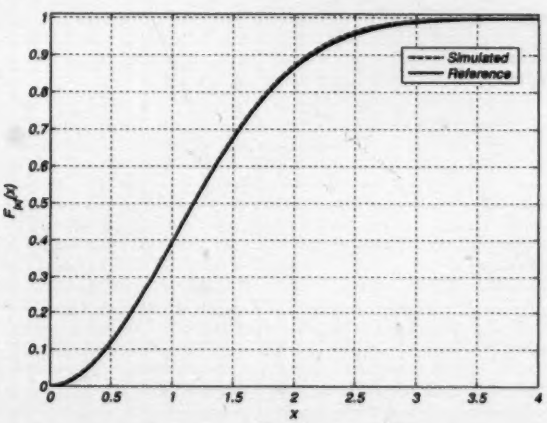

Figure 4.27: CDF of 10^7 fading variates generated by *Model II*.

Table 4.4: Implementation of the fading channel simulator on three different FPGAs.

Device family	I[a]	II	III
Max. clock freq. (MHz)	195.61	204.75	103.01
Output rate (MSamps/sec)	195	204	103
Slice utilization	2447 (9%)	2444 (5%)	1292 (1%)
Dedicated resource utilization	48 (9%)	48 (10%)	128 (72%)
Number of BRAMs	12 (3%)	12 (2%)	2%

[a]Design I was synthesized for a Xilinx Virtex4 XC4VSX55-11 FPGA. Design II was synthesized for a Xilinx Virtex2P XC2VP100-6 FPGA. Finally, Design III was synthesized for an Altera Stratix EP1S80F1508C6 FPGA.

tion language model and synthesized for the three typical FPGA devices given in Table 4.4. We used the PNG described in [97]. This PNG generates uniformly distributed 32-bit unsigned values between $(0, 1)$. The sine and cosine values used to calculate $\sin(\alpha_n[m])$ and $\cos(\alpha_n[m])$, respectively, were stored in four dual-port memories TBLROM12, TBLROM34, TBLROM56, and TBLROM78, each configured in 512×32 format. Eight dual-port cosine ROMs store cosine values used to calculate the in-phase and quadrature components $c_i[m]$ and $c_q[m]$. The hardware-based fading channel emulator design was adjusted carefully to achieve accurate fixed-point representations of the variables and also to minimize the computational resources. Specifically, the stochastic process θ was represented in 32-bit format, while $\phi[n]$ and $\varphi[n]$ used 10-bit precision. The values of $\sin(\alpha_n[m])$ and $\cos(\alpha_n[m])$ were represented in 12-bit format and the cosine values to calculate $c_i[m]$ and $c_q[m]$ were represented in 16-bit fixed-point format. The implementation of the fading channel emulator on a Xilinx Virtex2P XC2VP100-6 FPGA uses only 5% of the configurable slices, requires 48 dedicated 18×18 multipliers, and 12 Block RAMs. As shown in Table 4.4, the maximum sampling rate of the fading channel emulator on a Altera Stratix EP1S80F1508C6 FPGA is slower while utilizing only 1% of the configurable logic elements and uses 128 dedicated DSP blocks. We extracted the statistical properties of the generated fading variates and compared them against the software simulation results to successfully verify the accuracy of our hardware-based fading channel emulator. Figure 4.28 shows the layout of a $472,430\ \mu m^2$ semicustom integrated circuit implementation of the Rayleigh fading channel emulator designed in a 90-nm CMOS technology using a dual-threshold standard cell library. The core was targeted to operate at 500 MHz, generating 500 million 16-bit complex fading variables per second.

Figure 4.28: Layout of the 500 MHz semicustom fading channel variate generator.

4.7 Conclusions

To accurately generate Rayleigh faded envelopes that are correlated in time, but uncorrelated between processes, the quadrature components of fading variates must be uncorrelated, with each component having an autocorrelation given by a zeroth-order Bessel function of the first kind. Numerous algorithms with different computational complexities have been proposed in the literature to simulate a Rayleigh fading channel; however, most do not accurately reproduce the reference statistical properties of wireless propagation channels. Thus, the accuracy of the proposed models must be verified before using these models in a MC simulation scheme. While the SOS-based fading channel model is an efficient approach for hardware implementation, some of the proposed algorithms provide a close match with the theoretical statistical properties only when the statistics are averaged over an ensemble of fading channels. Hardware-based simulators permit several orders of magnitude faster performance evaluation over software-based simulators, significantly reducing the design time. The implemented SOS-based fading simulator uses only 1% of the Xilinx Virtex2P XC2VP100-6 FPGA and operates at 221 MHz, generating 211 million complex fading coefficients per second. The hardware-based fading channel simulator is 506 times faster than a software-based simulator written in C language running on a 3.4-GHz Pentium 4 (Xeon) with 2 GB memory.

Also, a novel technique for simulating Rayleigh fading channels with improved statistics was proposed. The proposed simulator was designed based on the sum-of-sinusoid Rayleigh fading models. A compact fixed-point implementation of the new emulator on

a single FPGA produces over 200 milion 16-bit Rayleigh fading variates per second. The ability to implement an entire fading channel simulator on a fraction of a single FPGA should be a significant improvement for the prototyping and verification of wireless systems.

Chapter 5

Filter-Based Fading Channel Simulators

Numerous real-time test cases must be applied to a new communication system design before shipping the products to market. Field tests in a mobile environment are expensive and may require permission from regulatory authorities. A further complication is that, due to the changeable nature of the mobile propagation environment, it is difficult to generate repeatable field test results. Instead, a multipath fading channel simulator allows the performance evaluation of mobile communication systems under controlled and repeatable conditions that would not normally be possible in actual field testing. When a fading channel simulator is used in the design and verification of wireless communication systems, it is important that the channel model represent all of the relevant behaviour and properties of a propagation environment as accurately as possible.

A complex Gaussian WSS process with the complex envelope $c(t) = c_i(t) + jc_q(t)$ has been commonly used to model the behaviour of multipath fading channels [113]. Under the common assumption of a two-dimensional isotropic scattering environment with an omnidirectional receiving antenna at the receiver [30], the PSD functions of $c_i(t)$ and $c_q(t)$ have the band-limited U-shaped form, the so-called Jakes PSD. Chapter 4 presented a well-known model for generating fading processes that approximate the Jakes power spectrum using Rice's sum of sinusoids (SOS) model, where a Gaussian process is modeled by the superposition of a finite number of weighted harmonic functions with random phases. This chapter considers an alternative technique, called the *filter-based* approach henceforth, for shaping the flat spectrum of uncorrelated Gaussian variates using a low-pass filter [114, 134, 139, 142]. This filter is often referred to as the *shaping filter* for it determines the power spectrum shape and the temporal correlation function of the fading process. The filter-

Table 5.1: Some commercially available fading simulators.

Model [a]	A	B	C	D	E
Number of channels	2	2	2	2	6
Number of paths	12	24	48	6	6
Max. Doppler (Hz)	800	2000	2400	-	340
Fading resolution (Hz)	0.1	0.01	0.05	-	1
Max. delay (ms)	200	2000	10	-	40
Time resolution (ns)	5	0.1	1	1	40

[a](A) Japan Radio Co. NJZ-1600B [144], (B) Spirent Communications SR5500 [143], (C) Agilent Technologies Inc. N5115A [6], (D) Rohde & Schwarz ABFS [7], (E) Ascom Ltd. SIMSTAR [138].

based approach can be customized to accurately provide the statistical properties required for simulating fading channels [23].

Due to the accuracy of this model for generating fading variates with reference statistical properties, many commercially available fading channel simulators [6, 7, 138, 143, 144] employ the filter-based technique. Typically they require complex hardware consisting of several circuit cards with multiple processors. For example, the NoiseCom MP-2500 Multipath Fading Emulator [145] consists of 11 circuit boards, not including the RF circuitry, cooling fans, or external computer interface for setting various parameters of a frequency-selective fading channel with up to 12 paths. Unfortunately these systems are rather bulky and costly. Some of the available fading channel emulators in the market are listed in Table 5.1 and are available at prices of between $24,000 to $500,000.

A software implementation of multipath fading channels using GPPs and DSPs is a more flexible and cost-effective scheme than such hardware-based commercial products. However, the implementation effects of nonlinear filters and nonlinear amplifiers are difficult to characterize analytically. Moreover, hardware-based simulators can verify the design at-speed, significantly reducing the simulation time compared to software-based testing schemes, and hence they reduce the time-to-market. The published fading channel simulators are commonly realized on heterogeneous architectures (usually consisting of GPPs, DSPs, FPGAs, etc.) to implement the computationally-intensive multi-rate signal processing algorithms of filter-based techniques [121, 123]. In these simulators, those portions of the simulator that are inefficiently simulated on a GPP can be off-loaded to a dedicated device, such as a FPGA or a DSP. For example, the fading simulator in [123] uses a floating point DSP combined with a FPGA to realize a frequency-selective channel simulator. The design in [121] uses two 32-bit floating-point DSP processors to implement a multipath fading simulator. Implementing a parameterizable fading channel simulator on a single FPGA

is a challenging task due to the relatively large computational complexity of the filter-based signal processing algorithms [123].

This chapter presents a novel design and implementation scheme for realizing a parameterized fading channel simulator on a homogenous architecture. Specifically, the new design is an accurate filter-based fading channel simulator on a single FPGA that is compact enough to be integrated along with many communication circuits of interest. To generate complex Gaussian variates with a U-shaped power spectral density function, the design utilizes an IIR spectrum shaping filter followed by multistage interpolators and low-pass IIR filters. In order to produce samples with accurate statistics and minimum hardware requirements, the required filters are designed in co-ordinated fashion. The new technique significantly alleviates the challenges of real-world testing of communication systems by introducing a fast and area-efficient FPGA implementation of the fading channel. Our fixed-point implementation of a Rayleigh fading channel simulator on an FPGA utilizes only 4% of the configurable slices, 20% of the dedicated multipliers and, 2% of the available memories on a Xilinx Virtex2P XC2VP100-6 FPGA, while generating 25 million fading variates per second. The parameterized mobile channel simulator can be reconfigured to accurately simulate a wide variety of different channel characteristics. We have also designed a flexible and compact filter processor (FP), called "Python", for efficiently implementing the shaping filter and interpolation low-pass filters (ILPFs) on the FPGAs. Python uses a simple and a very short instruction set to generate multiple sequences of fading variates for simulating wideband and MIMO channels.

The rest of this chapter is organized as follows. The digital filters structures and a precision analysis are discussed in section 5.1. An efficient method to generate correlated random sequences is presented in Section 5.2. Section 5.3 discusses the design constraints of the shaping and interpolator filters. Section 5.4 studies the major challenges in the design and hardware implementation of any filter-based Rayleigh fading channel simulator. Our proposed datapath for implementing a discrete-time fading channel simulator and implementation results are presented in section 5.5. The statistical properties of the generated fading variates are also verified. Section 5.6 presents the Python FP architecture and the implementation and statistical results of fading channel simulator. Finally, Section 5.7 makes some concluding remarks.

5.1 Digital Filter Structures and Quantization Error Effects

In digital signal processing, difference equations have been used extensively to model LTI systems. A difference equation has the general from [10]

$$\sum_{k=0}^{N} a[k]y[n-k] = \sum_{m=0}^{M} b[m]x[n-m]; \ \forall n \tag{5.1}$$

where $a[k]$ and $b[m]$ are constant coefficients (also called *gain factors*). If we scale the coefficients so that $a_0 = 1$ we obtain

$$y[n] = \sum_{m=0}^{M} b[m]\, x[n-m] - \sum_{k=1}^{N} a[k]\, y[n-k] \tag{5.2}$$

which clearly shows that the present output value $y[n]$ can be computed from the present input $x[n]$, $M \geq 0$ past input values and $N \geq 0$ past output values. In digital signal processing, Equation (5.2) is solved forward (i.e., for $n \geq 0$). Thus initial conditions on $x[n]$ and $y[n]$ must be determined for $-M \leq n \leq -1$ and $-N \leq n \leq -1$.

If the unit impulse response $h[m]$ LTI system (digital filter) is of finite duration (i.e., $h[n] = 0$ for $n_2 \leq n \leq n_1$), then the systems is called a *finite-duration impulse response* (FIR) filter [10]. A causal FIR filter can be expressed as

$$y[n] = \sum_{m=0}^{M} b[m]\, x[n-m]; \quad n \geq 0 \tag{5.3}$$

where $b[m] = h[m]$ for $m = 0, \cdots, M$, while all other $h[m]$'s are zero. An FIR filter is also sometimes called *non-recursive* or *moving average* (MA) *filter*. If the impulse response of an LTI system is of infinite duration, the system is called an *infinite-duration impulse response* (IIR) *filter*. The difference Equation (5.1) describes an IIR filter that has two parts where the following part

$$\sum_{k=0}^{N} a[k]y[n-k] = x[n]; \ \forall n \tag{5.4}$$

describes a recursive IIR filter with infinite duration in which the output $y[n]$ is recursively computed from its previously computed values and present input. Such a filter is called an *autoregressive* (AR) *filter*. The IIR filter in Equation (5.1) sometimes called an *autoregressive moving average* (ARMA) *filter* since it has both an AR part and an MA part.

An important subclass of causal LTI systems satisfy an N-th order linear constant-coefficient difference equation (CCDE) of the form

$$\sum_{k=0}^{N} a[k]\, y[n-k] = \sum_{k=0}^{M} b[k]\, x[n-k].$$

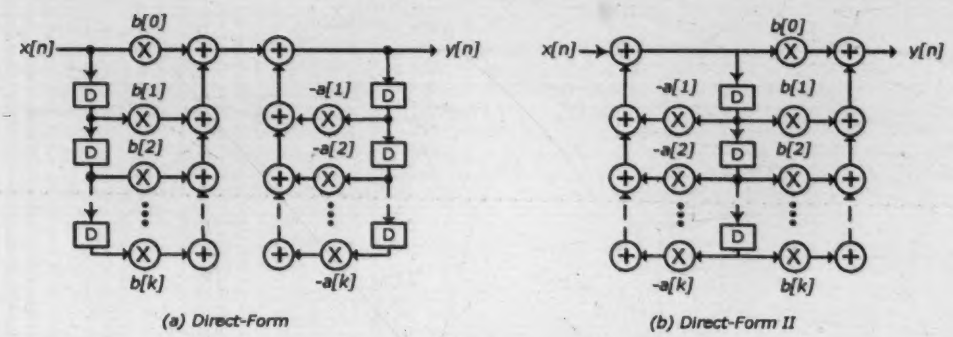

Figure 5.1: Direct-Form network structures.

It can be shown that the system function $H(z)$ can be represented as the rational function

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b[k]\, z^{-k}}{\sum_{k=0}^{N} a[k]\, z^{-k}} = \frac{\sum_{k=0}^{M} b[k]\, e^{-j\omega k}}{\sum_{k=0}^{N} a[k]\, e^{-j\omega k}} = \left(\frac{b[0]}{a[0]}\right) \frac{\prod_{k=1}^{M}(1 - c[k]\, z^{-1})}{\prod_{k=1}^{N}(1 - d[k]\, z^{-1})}$$

where z is the z-transform variable, $Y[z]$ and $X[z]$ are *feed-forward* and *feedback polynomials*, respectively, in z, and $H(z) = \sum_{n=0}^{\infty} h[n] z^{-n}$ is called the one-sided z-transform of the *rational transfer function* (RTF) of the LTI system. Each of the $(1 - c[k]z^{-1})$ factors contributes a *zero* (i.e., a root of the numerator) at $z = c[k]$ and a *pole* (i.e., a root of the denominator) at $z = 0$; similarly, each $(1 - d[k]z^{-1})$ contributes a pole at $z = d[k]$ and zero at $z = 0$. Thus the $b[k]$'s are the filter's feedforward coefficients corresponding to the zeros of the filter, and the $a[k]$'s are the filter's feedback coefficients corresponding to the poles of the filter.

It is shown in [10] that any sequence that can be represented as a sum of exponentials, and can also be represented by a rational z transform. $H(z)$ is sometimes represented as

$$H(z) = \frac{\sum_{m=0}^{M} b[m] z^{-k}}{1 + \sum_{k=1}^{N} a[k] z^{-k}}$$

by taking the z-transform of the both sides of the difference equation (5.2). For convenience in the notation we assume equal feed-forward and feedback orders, i.e., $M = N$. A block diagram of the corresponding filter implementation (called the *Direct-Form*) is shown in Figure 5.1(a). This flowgraph clearly separates the structure into a section for the zeros on the left and a section for the poles on the right. If you split the Direct-Form shown in Figure 5.1 at the summation point and swap the two halves, so that the feedback half (the poles) comes first, then the two pairs can be merged, yielding a more compact configuration than the Direct-Form called the *Direct-Form II*, as shown in Figure 5.1(b). For a second-order

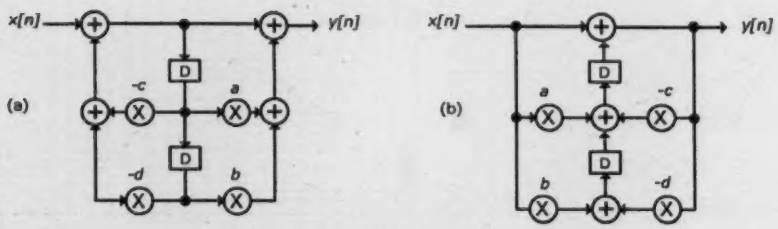

Figure 5.2: (a) Direct-Form II. (b) Direct-Form I.

(i.e., two poles and two zeros) IIR filter, the flowgraph, shown in Figure 5.2(a), is called a canonic section or a *biquad*. Note that in a Direct-Form II, the states (delayed samples) are neither the input nor the output samples but are instead derived intermediate values. It is shown that a K-th order IIR filter can be designed as a cascade of $L = \lfloor (K+1)/2 \rfloor$ such biquads [10]. Then, the discrete transfer function can be written as

$$H(z) = G \prod_{k=1}^{L} \frac{1 + a[k]z^{-1} + b[k]z^{-2}}{1 + c[k]z^{-1} + d[k]z^{-2}} \tag{5.5}$$

where when K is odd, and $b[L]$ and $d[L]$ are zero. Biquads come in different forms. A transposed network (the directions of all branches in the flowgraph are reversed, with all branch nodes in the original network becoming summation nodes in the transposed network and vice versa), shown in Figure 5.2(b), is called *transposed Direct-Form II* or *Direct-Form I*. It provides the same system function using two adders and a summation node (with three inputs), while in a biquad, four two input adders are utilized.

Both the throughput and latency of a digital implementation are affected substantially by the choice of the network structure. One practical drawback of the recursive filter structures in the IIR filter structures is that they set an upper bound on the sample rate. For example, consider the cascade Direct-Form II structure shown in Figure 5.3. The direct mapping of a cascade structure onto hardware has a relatively long combinational propagation delay to the output. This path, as shown with a thick arrow in Figure 5.3, contains $2L + 1$ adders, one multiplier and one register. In order to achieve higher performance (and hence a shorter clock cycle), a pipelined cascade architecture can be created by adding one register at the output of each biquad stage. Then, the critical path delay is reduced to the delay of a register, one multiplier and three two-input adders. However even this delay can be significant when realizing a digital filter on a FPGA.

One important decision is to determine the number of bits required to represent the constant coefficients and internal signals in the filter. It is well-known that as the order of

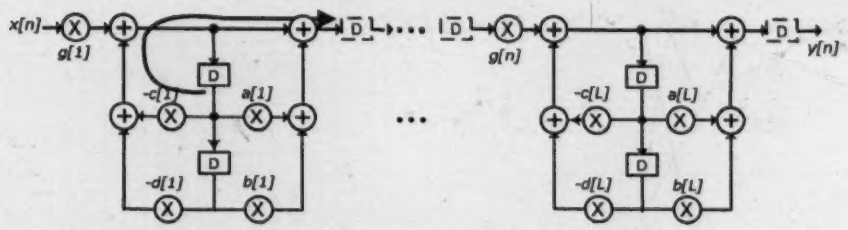

Figure 5.3: Cascade realization of the spectral shaping filter.

the polynomial increases, so too does the sensitivity of the obtained roots of a polynomial to the accuracy of its coefficients [10]. Also, for a narrowband filter with a high sampling rate, the zeros and poles tend to be crowded near the unit circle. If the poles reside on the unit circle or outside of it, the filter becomes unstable [10]. As the coefficients are quantized for a fixed-point platform, due to the rounding or truncation of values, the quantization error can be fed back in the filter, successively magnifying the total error and causing instability. Even if the filter stays stable, the poles can be displaced significantly from their design locations by the quantization of the coefficients, and thus the target specification will not be achieved. Therefore, a careful analysis is required to ensure adequate precision in the coefficients. Since FPGAs, unlike fixed register size DSPs, allow custom bit widths, the coefficient representation can be different than the internal signal representation, providing a greater flexibility. The internal signal bit widths then can be determined by calculating theoretical bounds on the dynamic ranges of the signals, and on the maximum output errors introduced by truncation in the fixed-point representation. The lower and upper bound values of each signal state how many bits are required at any point in the computation in order to minimize the probability of overflow/underflow while guaranteeing a prescribed degree of accuracy at the filter output.

The fixed-point implementation of digital filters is discussed in many textbooks and in the published literature. For example, it is shown that when the denominator coefficients of the second-order filter are perturbed by Δc and Δd (i.e., perturbation from its ideal infinite precision value), respectively, the poles of the transfer function, p_1 and p_2, will also be perturbed by

$$\Delta p_1 = \frac{p_1 \Delta c + \Delta d}{p_2 - p_1} \text{ and } \Delta p_2 = \frac{p_2 \Delta c + \Delta d}{p_1 - p_2} \tag{5.6}$$

respectively [146]. A similar relation holds for perturbations affecting the zeros of the transfer function. According to the given maximum allowable percent change in the pole

location relative to the unit circle ξ_p, one can obtain the number of fraction bits for c and d in (5.5) WF as $WF = \lceil -\log_2(\xi_p|1 - |p_i||.|p_2 - p_1|)\rceil + 1$ for $i = 1, 2$ [146]. While the error analysis of fixed-point implementations of digital filters has also been addressed in a large number of papers and textbooks [10, 147], the software package Matlab Filter Design Toolbox [148] offers numerous libraries for the analysis of finite wordlength impact on the numerical stability of the designed filters.

Choosing the computational structure depends on the order and specification of the filter, the desired sampling rate, the susceptibility to quantization error, and the available configurable resources on the FPGA. While other network structures, such as the coupled form [10, 147], have also been proposed to reduce the sensitivity to inaccuracies in the coefficients and to signal quantization, it is shown that the cascade form is a robust structure under quantization when compared with many other schemes.

5.2 Generation of Correlated Random Sequences

If the input to an LTI system is a random process $X(t)$, then the output $Y(t)$ is a random process [14] given by

$$Y(t) = \int_{-\infty}^{\infty} h(\tau)X(t-\tau)\mathrm{d}\tau = \int_{-\infty}^{\infty} h(t-\tau)X(\tau)\mathrm{d}\tau.$$

Similarly, for a discrete-time RP $X[n]$,

$$Y[n] = h[n] * X[n] = \sum_{j=-\infty}^{\infty} h[j]X[n-j] = \sum_{j=-\infty}^{\infty} h[n-j]X[j]$$

and

$$H[f] = \sum_{i=-\infty}^{\infty} h[i]\, e^{-j2\pi fi}.$$

If $X(t)$ is a WSS process, then $Y(t)$ is also WSS and the corresponding PSD is

$$\mathrm{G}_Y(f) = \mathrm{G}_X(f)|H(f)|^2. \tag{5.7}$$

Equation (5.7) states that the output PSD equals the input PSD multiplied by the squared magnitude of the transfer function of the filter. In many applications, such as frequency-flat fading simulators, we need to generate one (or more) random sequences with a particular correlation function. For example, for the representation of phase noise in a communication system, a Gaussian process with an arbitrary PSD (which might be defined based on empirical measurements) is often required [11]. When $\mathrm{R}(\tau) \neq 0$ for $\tau = kT_s$, $k = \pm 1, \pm 2, \cdots$,

the RP is correlated at multiples of the sampling intervals. We note that if $X(t)$ is a white noise process with the flat PSD $\mathrm{G}_X(f) = N_o/2$, then the transfer function completely determines the shape of the PSD of the output process as

$$\mathrm{G}_Y(f) = N_o/2|H(f)|^2. \tag{5.8}$$

Equation (5.8) shows that an *uncorrelated* Gaussian sequence can be transformed into a *correlated* Gaussian sequence through a linear filter that preserves the Gaussian distribution but alters the correlation properties. This technique can be efficiently used for generating WSS processes with arbitrary output PSD (corresponding to a desired output correlation). Hence, a filter with the transfer function

$$H(f) = \sqrt{\frac{2}{N_o}\mathrm{G}_Y(f)} \tag{5.9}$$

can be designed in the frequency domain to transform a white noise process into a coloured random sequence if $H(f)$ can be represented as a RTF, i.e., the ratio of two polynomials as discussed in Section 5.1. An ARMA filter can be utilized to implement $H(f)$ in (5.9) as a RTF for generating random sequences with arbitrary PSD [13]. A very important point to note is that the phase response of the filter is not important since it does not affect the output PSD [13].

In practical applications, such as frequency-selective fading channel simulators and MIMO channel simulators, we need to generate more than one correlated random sequence. Multiple random processes can be represented as a vector-valued RP where each vector component is a RP with a particular ACF (*temporal correlation* or correlation along time axis) and the correlation between vector components represents a correlation in a different dimension (referred to as *spatial correlation*) [13]. Assume that we want to generate sampled values of m zero-mean Gaussian random processes $Y_1(t), Y_2(t), \cdots, Y_m(t)$, denoted as a vector-valued discrete-process $\mathbf{Y}[k] = \left[Y_1[k], Y_2[k], \cdots, Y_m[k]\right]^T$, with the same arbitrary temporal correlation $\mathrm{R}[n]$ (or equivalently, the same PSD). Further assume that we want to have some arbitrary correlation between processes such as $\mathrm{R}_{Y_iY_j}[n] = \mathrm{E}\left[Y_i[k]Y_j^*[k+n]\right] = \sigma_{ij}\mathrm{R}[n]$, where σ_{ij} is the covariance between the components of the process at a given instant. To generate a sequence of Gaussian vectors $\mathbf{Y}[k]$ that are correlated in time and space, we can transform a sequence of uncorrelated (and hence independent) Gaussian vectors $\mathbf{X}[k]$ into a time-correlated Gaussian sequence $\mathbf{Y}[k]$, with each sequence generated using an ARMA filter, and then transform spatially-uncorrelated com-

ponents of $\mathbf{Y}[k]$ into spatially-correlated components using a (memoryless) linear transformation [13].

5.3 Constraints on Filter Design

As noted earlier, it is common to model the behavior of multipath fading channels as a complex Gaussian WSS process $c(t) = c_i(t) + jc_q(t)$ [113]. In a two-dimensional isotropic scattering environment with an omnidirectional receiving antenna at the receiver [30], the ACF associated with either $c_i(t)$ or $c_q(t)$ is given by $\mathrm{R}_{c_i,c_i}(\tau) = \mathrm{R}_{c_q,c_q}(\tau) = \mathcal{J}_0(2\pi f_D \tau)$. The PSD functions of $c_i(t)$ and $c_q(t)$, denoted by $\mathrm{G}_{c_i}(f)$ and $\mathrm{G}_{c_q}(f)$ respectively, can be written as

$$\mathrm{G}_{c_i}(f) = \mathrm{G}_{c_q}(f) = \begin{cases} \frac{\sigma_i}{\pi f_D \sqrt{1-(f/f_D)^2}} & |f| < f_D \\ 0 & |f| \geq f_D \end{cases} \tag{5.10}$$

where σ_i is the variance of c_i.

In order to generate the in-phase and quadrature components of fading variates with a particular correlation between variates, as discussed in Section 5.2, we begin with two independent, zero-mean, white Gaussian random variables $n_i(t)$ and $n_q(t)$ with identical variance. A linear filtering operation on the complex Gaussian samples with flat PSD, $n_i(t)$ and $n_q(t)$, yields samples that also have a Gaussian distribution, with spectrum $\mathrm{G}_{out}(f) = \mathrm{G}_{in}(f)\,|H(f)|^2$, where $\mathrm{G}_{in}(f)$ is the spectrum of the input samples and $|H(f)|^2$ is the squared magnitude response of the shaping filter. As described in Chapter 4, the theoretical spectral density of the complex envelope of the signal received by an omnidirectional antenna in a Rayleigh fading wireless channel is given by the Jakes PSD [13]. A shaping filter can be designed with a frequency response equal to the square root of the PSD of the desired fading process (i.e., $\sqrt{\mathrm{G}_{c_i}(f)}$). A correlated Rayleigh process can then be generated by combining the two filtered processes in quadrature.

Similarly, to generate correlated Rayleigh variates, the Gaussian-distributed in-phase and quadrature components can be spectrally shaped by multiplying the frequency domain Gaussian components by $\sqrt{\mathrm{G}_{c_i}(f)}$. Then an inverse fast Fourier transform (IFFT) can be applied to the resulting discrete spectrum to obtain time series data [114, 134]. The resulting series is still Gaussian by virtue of the linearity of the IFFT, and it has the desired Jakes spectrum. The pseudo-code of this approach is given in Algorithm 6 [114]. The IFFT has a computational complexity of $O(K \log K)$ operations, where K is the number of time-domain sampled Rayleigh channel coefficients. One major disadvantage of the IFFT

Algorithm 6 Modified Smith's algorithm to generate discrete-time samples of correlated Rayleigh fading process.

1: Specify the value of N equally spaced points of $\sqrt{\mathrm{G}_{c_i}(f)}$ with the frequency spacing between adjacent spectral lines as $\Delta f = 2f_D/(N-1)$. Thus the time duration of waveform is $\tau = 1/\Delta f$.
2: Generate $N/2$ complex Gaussian random variates for each of the $N/2$ positive frequency components of $\sqrt{\mathrm{G}_{c_i}(f)}$.
3: Construct the negative components of the noise source by conjugating positive frequency values. This will yield a real waveform.
4: Multiply the N noise points of the inphase and quadrature components of the noise sources by the discrete frequency representation of $\sqrt{\mathrm{G}_{c_i}(f)}$.
5: Perform IFFT on the resulting frequency domain signal from inphase and quadrature branches to form complex time samples.

method is its block-oriented nature, which requires all channel coefficients to be generated and stored before the data is sent through the channel. This implies significant memory requirements and precludes unbounded continuous transmission, which is usually preferred in long running characterization applications.

The reciprocal square root in Equation (5.10) is an irrational function [139], which cannot be implemented exactly in hardware, so it is common to use a rational approximation of the Jakes PSD. To provide spectral shaping for a rational implementation, we used transformation-based filter designs [121, 123]. In wireless communication channels, the Doppler frequency is typically much smaller than the sampling rate. This greatly reduces the required bandwidth of the spectral shaping filter. For example, consider the digital cellular system DSC1800 (GSM1800), which operates at $f_c = 1.8$ GHz. If the mobile receiver has a maximum speed of $v = 300$ kmph, then the maximum Doppler frequency would be $f_D = f_c \times (v/c) = 500$ Hz, where c is the speed of light. If the signal is sampled at $R_s = T_s^{-1} = 10$ MHz, then the normalized Doppler frequency would be $f_D T_s = 0.00005$. However, a symbol-rate design of an extremely narrow-band digital filter may run into numerical problems [147]. Instead, it would be more stable and computationally-efficient to design the PSD shaping filter for a lower sampling frequency R_{ch}, and then increase the sampling frequency using interpolation to achieve the target symbol rate. The sampling rate of the filter must be increased by an *interpolation factor* $I = \lceil \frac{R_s}{R_{ch}} \rceil$ to be compatible with the signal sampling rate, where R_{ch} is the channel sampling rate used to design the shaping filter. After interpolation, a low-pass filter is utilized to eliminate the replicas of signal introduced in the frequency domain by interpolation. For a practical mobile system, factor I can be large and thus the complexity of the real-time interpolation filter can

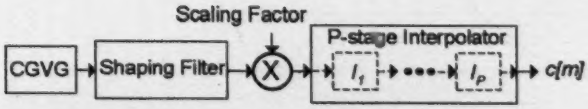

Figure 5.4: Architecture of a filter-based fading channel simulator.

be large. To reduce the computational complexity of the low-pass filters, interpolation is usually accomplished using a multi-stage interpolator design.

The functional structure of the filter-based fading channel is represented in Figure 5.4. The generated samples from the complex Gaussian variate generator (CGVG) are passed through a shaping filter and multiplied by a scaling factor to normalize the power of the final resulting channel variate $c[m]$. Then the sampling rate of generated fading variates is increased using a P-stage interpolator.

The fading channel simulators in [114, 127, 149, 150] use FIR filters as the shaping filter while the designs in [120, 123, 151, 152] used IIR filters. Several important points should be considered when implementing fading channel simulators using FIR and IIR filters on hardware platforms:

- The degree of the FIR filter is related to the time span of the truncated signal held in the filter and inversely proportional to the Doppler frequency. Specifically, implementation of an extremely narrow-band digital filter with a sharp cutoff and very large attenuation in the stop-band requires a large-order FIR filter [13, 152]. Meeting the same specifications with an IIR filter typically requires fewer hardware resources than an FIR filter. In fact, utilizing both feedforward and feedback polynomials in an IIR filter permits steeper frequency roll-offs to be implemented for a given filter order than an FIR filter [147]. Thus, rather than designing a high-order FIR filter for an extremely small $f_D T_s$ provided (e.g., 10^{-5}), an IIR filter can be designed with a smaller order and the resulting filter is less computationally-expensive for a larger maximum Doppler rate (e.g., $f_D T_s = 0.1$).

We designed and implemented bandlimited Jakes spectral shaping filters using an IIR filter to closely approximate $G_{c_i}(f)$ in the passband while providing large suppression in the stopband. An elliptic filter is an efficient candidate that has an especially sharp transition from the passband to the stopband for a given order [147]. A discrete-time elliptic filter can be designed using the cascade Direct-Form II second-order sections (biquads) structure that is more robust under quantization than the Direct-Form structure [147] (e.g., a cascade of four biquads is used in [28] and a cascade of seven biquads is used in [139]). As shown in

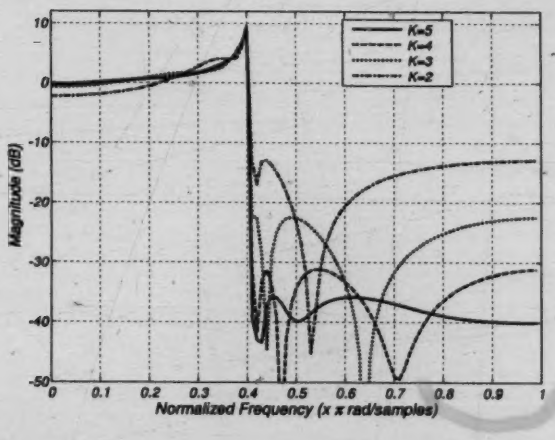

Figure 5.5: The magnitude responses of elliptic filters composed of $K = 2, 3, 4, 5$ cascaded biquads with $f_D T_s = 0.2$.

Figure 5.5, for a fixed maximum discrete Doppler rate $f_D T_s = 0.2$, the magnitude response of filters designed with $K \geq 4$ cascaded Direct-Form II second-order sections (biquads) closely matches the ideal response in the passband and has a steep roll-off in the transition to the stopband. In this case, the PSD of the filter with $K = 5$ has at least 8 dB of peaking above $G_{c_i}(0)$ and has rolled off at least 35 dB with respect to $G_{c_i}(0)$ for $R_{ch} > 2f_D$.

• An FIR filter has no feedback and is thus inherently stable. However, as the coefficients are quantized in any fixed-point implementation, the resulting numerical error is fed back in the IIR filter, possibly causing instability. Moreover, such effects can cause significant deviations from the expected response. To make sure that the filters are stable under quantization effects, we have designed the filters in fixed-point format using Filter Design Toolbox by Matlab [148] that offers bit-true implementations of second-order sections with section scaling and reordering to obtain maximum accuracy. Word lengths for different internal signals and coefficients are set to the values verified by fixed-point simulation of the filters. Dynamic range and round-off noise analysis provided by the toolbox will also assure that the chosen fixed-point representation will avoid numerical instability.

• For a fixed signal sampling rate of R_s, the lower the channel sampling rate, the larger must be the interpolation factor. For a high R_{ch}, the filter order will increase and the filter will become more sensitive to quantization. There is therefore a tradeoff between the computational-complexity and numerical stability of the shaping filter and the computation requirements of the interpolation filter. There are many possible combinations of shaping filter designs and interpolator implementations. It is reasonable to design the shaping filter

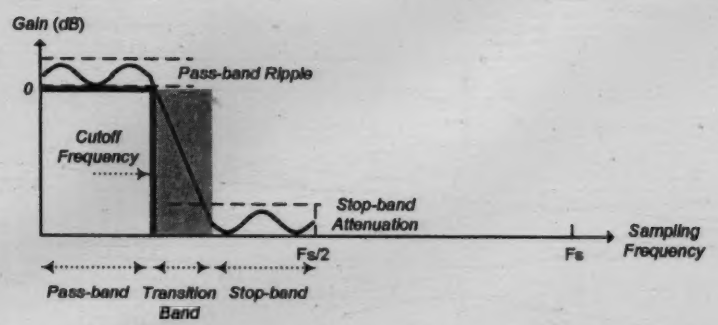

Figure 5.6: Ideal low-pass filter specification.

with a small R_{ch} (compared to R_s), which provides stable operation for a fixed-point realization with acceptable computational complexity. Then the interpolator can be designed and implemented efficiently using a variable polyphase filter, to accommodate different Doppler rates, with a windowed $sinc(\cdot)$ function impulse response [13]. Note that when the digital filter is designed at a fixed Doppler rate $f_D T_s$ to provide sufficiently accurate frequency response, then the structure in Figure 5.4 is not flexible enough to produce a Rayleigh fading waveform with an arbitrary discrete Doppler rate. For an irrational value of $f_D T_s$, if the IIR spectral shaping filter is designed for a fixed discrete maximum Doppler rate, then the interpolator must re-sample the process so that rational fractions of the desired rate $f_D T_s$ can be approximated. The irrational Doppler rate factor may not be a serious limitation when simulating a fading channel because in performance verification, obtaining high accuracy in the Doppler rate is not usually of primary importance [139].

Filters are usually specified using only a few parameters such as passband, stopband, and the tolerance allowed within these bands [10]. As shown in Figure 5.6, the *filter bandwidth* is the width between two frequencies that define the upper and lower edges of the pass-band. The *transition band* is an interval of frequencies where a filter characteristic changes from one kind of behaviour to another (for example, from a *pass-band* to *stop-band*). The behaviour of a filter's response that approximates a desired characteristic by being alternatively greater and less than the desired response is called the *ripple*. Pass-band ripple also called in-band ripple and stop-band ripple is also called out-of-band ripple. There are different standard filters that are analytically described by the type of polynomial used in $H(z)$ such as the Chebyshev, inverse Chebyshev, and elliptic filters [10]. To design a discrete-time digital filter, various (iterative) optimization techniques can be utilized to minimize the mean-square deviation from a desired frequency-domain characteristic.

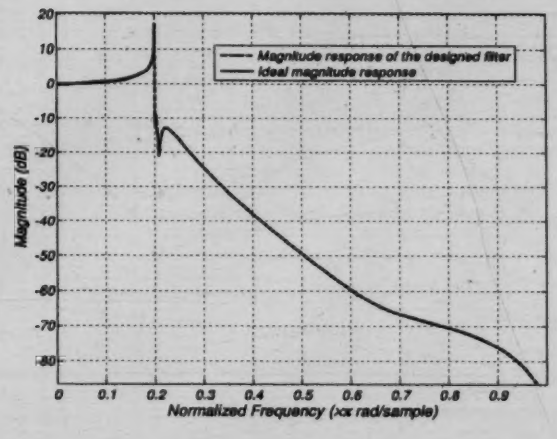

Figure 5.7: Magnitude response of the elliptic shaping filter.

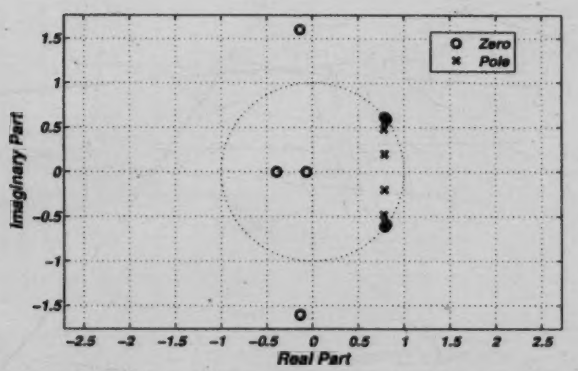

Figure 5.8: Zero/pole plot of the 32-bit quantized shaping filter.

In order to implement the Jakes PSD, we used an IIR elliptic filter that has the sharpest transition from the pass-band to the stop-band for a given order among the different classical IIR filters [147]. We also followed a typical assumption that the Doppler spectrum for mobile speeds of interest is less than 2 KHz wide [30]. We used 10 samples/period of the highest frequency of an analog signal [13], so $R_{ch} = 10f_D = 20$ Ksamples/sec. For a fixed maximum normalized Doppler rate $f_D T_s = 0.1$, we used $K = 5$ cascaded biquads to realize the Jakes PSD. As shown in Figure 5.7, the magnitude response of the resulting elliptic shaping filter matches the ideal response in the pass-band and has a sharp cutoff. Figure 5.8 shows the zero/pole plot of the 32-bit quantized shaping filter.

An important point to note in Figure 5.7 is that if the stop-band attenuation of the

shaping filter is not great enough, then the out-of-band noise passed through the filter can impact the accuracy of the statistics of the generated fading variates. Since designing a narrow-band filter with a sharp cutoff and large attenuation leads to a high order (and hence greater computational requirements), to obtain the closest approximation to the desired frequency response with a relatively small filter order, we only minimized the approximation error in the passband of the spectral shaping filter. The low-pass filters that follow utilized in the next stages are then designed with extra attenuation over the transition region and the stop-band.

In order to design a P-stage interpolator, we made the reasonable assumption that the WSS channel model can be used for the urban mobile radio channel over bandwidths of up to 10 MHz [21, 139]. The interpolation factor of our designed system can therefore be up to $I = \lceil R_s/F_{ch} \rceil = 500$. Typically a multi-stage interpolation scheme using FIR filters is employed [121, 123]. However, when the interpolation factor I is large, we found that this approach may not be efficient enough to be implemented on resource-limited FPGAs. As an alternative, we propose to use low-pass IIR filters with smaller degrees that provide almost linear phase response in the pass-band as ILPFs. A two-stage interpolator can be designed using two low-pass Chebyshev Type II IIR filters, one for $I_1 = 5$ and one for $I_2 = 100$. The designed ILPFs have a maximum of 0.01 dB attenuation in the passband and a minimum of 100 dB attenuation in the stopband. Figure 5.9 plots the magnitude response of the first ILPF designed using $P_1 = 5$ cascaded biquads. Figuer 5.9 also compares the response of this IIR filter with an equiripple FIR filter designed with the same parameters. The second ILPF was designed using an IIR filter with $P_2 = 2$ biquads where the corresponding equiripple FIR filter is of order 1067. Clearly, designing the ILPFs using IIR filters leads to a significantly smaller order, and this is much more computationally efficient. Figure 5.10 plots the phase response of the first ILPF. As shown in Figure 5.10, the designed ILPF provides almost linear phase in the pass-band (0 to 2 KHZ).

5.4 Filter Design for Fading Channel Simulators

We introduce our new general design method by means of an example. Assuming that the maximum Doppler frequency is $f_D = 4$ KHz and the target sampling rate is $R_s = T_s^{-1} =$ 10 MHz. To decrease the filter orders, we will design the shaping filter at the sampling rate $R_1 = T_1^{-1} = 20$ KHz, which is 2.5 times the Nyquist frequency. Later, the sampling frequency will be increased to $R_2 = T_2^{-1} = 100$ KHz and $R_s = 10$ MHz with $I_1 = 5$ and

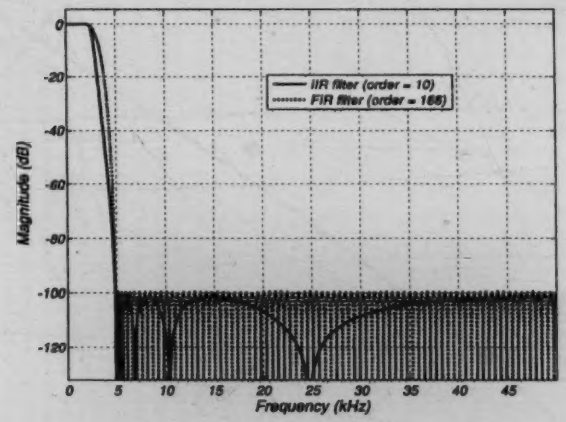

Figure 5.9: Magnitude response of the first stage IIR low-pass interpolation filter.

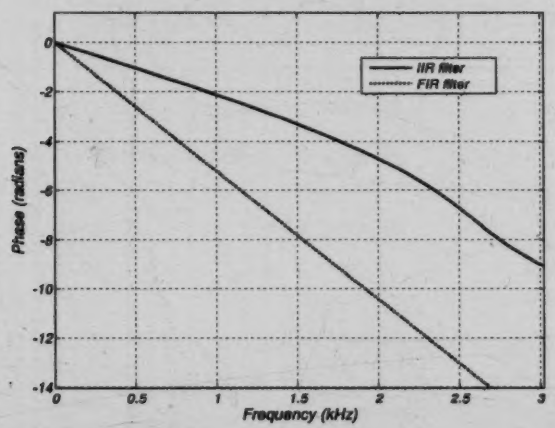

Figure 5.10: Phase response of the IIR low-pass filters for interpolation.

$I_2 = 100$ times interpolation, respectively.

The cascade of filters is designed in two steps. In the first step, the required filters are designed to meet the desired characteristics, and then in a second step, the filters are optimized to reduce their hardware complexity. Since no limitation is imposed on the phase of the target signal and considering the greater computational efficiency of IIR filters, we decided to approximate the PSD in (5.10) with a rational filter using the least pth-norm approximation [153]. This was achieved by utilizing the MATLAB function iirlpnorm [148]. This function allows one to weight the allowable error over different frequency ranges. Since the signal has a narrow bandwidth, we chose to utilize low-order IIR interpolation low-pass filters in the interpolation stages. Specifically, we used inverse-Chebychev low-

pass filters for interpolation.

In the next step, the filters are changed to be more efficiently mapped onto the hardware. Since the shaping and first stage interpolation low-pass filters operate at relatively low sampling rates, as we explain further in the next section, it is advantageous to reuse hardware to implement both filters. Employing the second-order section (sos) form of the IIR filters simplifies hardware sharing. In fact a shared biquad can perform the processing of different sos parts in the shaping and the first-stage ILPF. However, to simplify the control unit, as will be discussed more in Section 5.5, it is important for the two filters to have the same number of sos's.

To minimize the total number of sos's in the shaping and the first-stage ILPF, they should be optimized *together* to meet the design requirements. In a "discrete" design, the shaping filter should ideally have a frequency response matched to (5.10) in the range $|f| < f_D = 4$ KHz and a zero response elsewhere. Also, the first ILPF should ideally pass the signal within the range $|f| < 10$ KHz (recall that $R_1 = 20$ KHz). However, with finite out-of-band attenuation, the desired statistics of the target Rayleigh fading samples might not be met. In particular, the LCR of the envelope of the fading samples is sensitive to the out-of-band attenuation. We decided to design these two filters together to minimize the error in the pass-band $|f| <4$ KHz while maximizing the attenuation in the stop-band $4<|f|<50$ KHz. Here we used the weights computed by iirlpnorm and the low-pass filter parameters as the search variables for the filter design algorithm. Figure 5.11 shows the frequency response of the resulting shaping filter with $K = 6$ second-order sections. As this figure shows, the shaping filter provides a frequency response that closely matches the desired response over the pass-band and up to frequencies just inside the stop-band. However, at higher frequencies the attenuation is somewhat less. Figure 5.12 shows that the first-stage ILPF provides at least 65 dB attenuation over frequencies where the shaping filter might not provide adequate attenuation. Therefore, the ILPF, not only attenuates the out-of-band signals, it can also help the shaping filter to provide more accurate samples.

The second-stage ILPF is designed using a similar technique. However, there is no constraint on the length of this filter. This allows us to use this filter to further increase the attenuation over the stop-band, where the shaping filter and the first-stage ILPF may not provide enough attenuation. However, since the second-stage ILPF runs at the higher frequency, minimizing the filter order significantly decreases the required processing time.

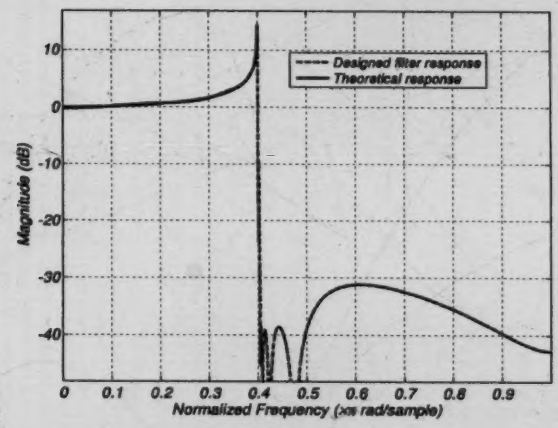

Figure 5.11: Magnitude response of the elliptic shaping filter.

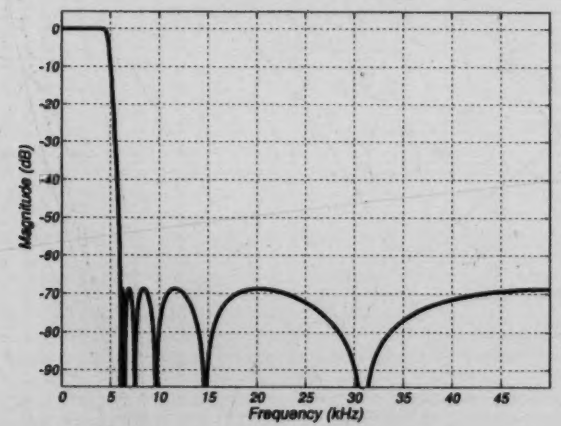

Figure 5.12: Magnitude response of the first stage IIR low-pass interpolation filter.

5.5 Implementation and Statistical Verification

When implementing the fading channel emulator in fixed-point arithmetic, the stability of the designed IIR filters after quantization is ensured by keeping all poles within the unit circle in the z-plane. Also, to maintain accuracy, the fixed-point format of the intermediate signals is chosen based on the numerical studies for different precisions that determine the impact on the statistical properties of the generated fading variates. A 32-bit fixed-point format was found to give sufficient accuracy.

The input to the filter chain is generated using GVGs. Realizing the fading channel simulator on an FPGA requires two GVGs (for the real and imaginary components), K cascaded biquads and $10K$ multipliers to implement the shaping filter, and P cascaded

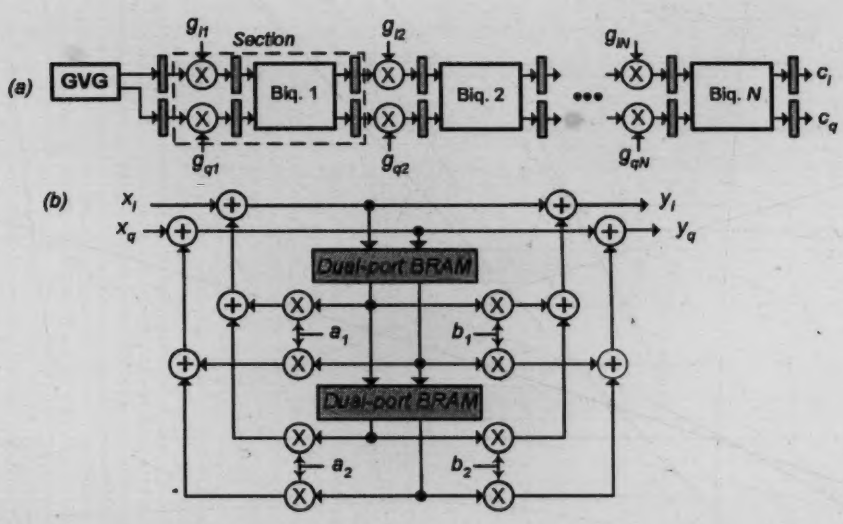

Figure 5.13: (a) The structure of cascading Gaussian variate generator and $N = K + P$ second-order sections. (b) Biquad datapath.

biquads and $10P$ multipliers to implement the ILPFs, as shown in Figure 5.13(a), where $N = K + P$. The biquad datapath is shown in Figure 5.13(b), where four registers are stored in two on-chip dual-port BlockRAM memories.

As the order of the filters is increased for greater accuracy, it becomes more challenging to realize the structure in Figure 5.13(a) with high-precision arithmetic components on resource-constrained FPGAs. For example, for a 32-bit realization, a section in Figure 5.13(a) requires 855 configurable slices and utilizes 9% of the dedicated 18×18-bit multipliers available on a Xilinx Virtex2P XC2VP100-6 FPGA. These results confirm that the maximum number of sections shown in Figure 5.13(a) that can be implemented on a large FPGA is only around 11 due to the relatively large number of high-precision arithmetic units required by each second-order section. Consequently, the direct instantiation of cascaded sections might be impractical for higher order filters (i.e., for smaller values of $f_D T_s$) or might not be efficient and flexible enough to implement variable sampling rates (e.g., for larger interpolation factors).

One widely-used solution is to utilize a heterogeneous architecture, usually consisting of general-purpose processors, DSPs, FPGAs, etc. [121,123]. The two main reasons are: (I) a direct implementation of a parameterizable Rayleigh fading channel simulator may not fit into one FPGA; and (II) the fading channel process usually varies much more slower than the signal transmission rate. This implies that the GVGs and the shaping filter can be up-

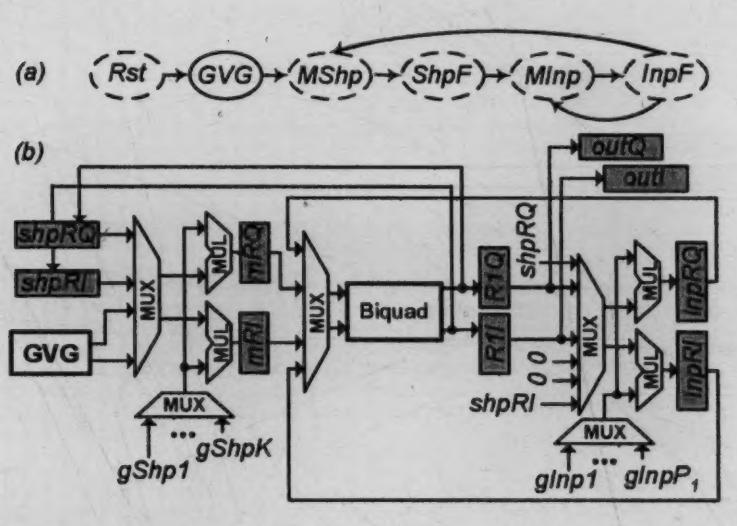

Figure 5.14: (a) Control data flow graph and (b) the datapath of the shaping filter and the first-stage low-pass interpolation filter.

dated at a much lower rate than the sampling rate of the system signal. Thus, it is reasonable to implement the GVGs and the shaping filter on a DSP, and the interpolator (or at least the last stage of a multi-stage interpolator) on a FPGA. To obtain the maximum performance with a minimum of FPGA resources, portions of the spectral shaping filter are time-shared with the ILPFs. Resource sharing of independent operations of the spectral shaping filter and ILPFs offers significant resource saving in the implementation of a computationally-intensive fading channel emulator. The throughput of a fading simulator depends on the binding of the second-order sections to the shared resources. It was found that the sampling rate of the hardware-based digital filters is high enough that the throughput reduction of the time-multiplexed scheme, compared to the direct instantiation approach, does not impact the maximum target sampling rate.

Due to the slow variation of samples at the shaping filter compared to the fast operating rate of biquads implemented on an FPGA, we implemented the shaping filter and the first ILPF using one shared biquad. The second ILPF is mapped to a separate set of configurable resources to achieve the target sampling rate. Figure 5.14(a) shows the control data flow graph (CDFG) that generates appropriate control sequence for the datapath (shaping filter and first ILPF) shown in Figure 5.14(a). The state machine controller starts in the reset state *"Rst"*, where the *"gShp1"* to *"gShpK"* registers are initialized with the K scaling factors of the spectral shaping filter, and registers *"gInp1"* to *"gInpP"* are initialized with the P_1 scaling

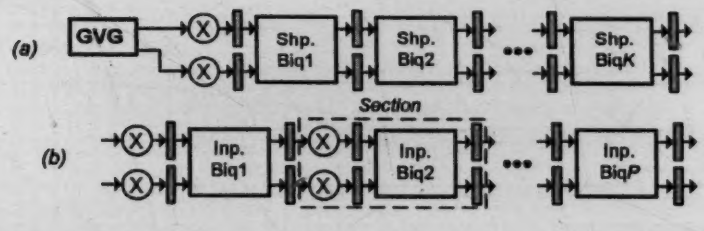

Figure 5.15: (a) The structure of cascading Gaussian variate generator and the shaping elliptic IIR filter. (b) The IIR Chebyshev low-pass filter structure designed using cascading biquads.

factors of the first ILPF. Intermediate registers, such as *"shpRI"* and *"shpRQ"*, are cleared to zero. In the next state, *"GVG"*, two Gaussian variates are generated. State *"MShp"* denotes the multiplier state of the shaping filter and state *"MInp"* denotes the multiplier state of the first-stage interpolator. At the *"ShpF"* (*"InpF"*) state, one of the K (P_1) sections of the shaping filter (first-stage ILPF) is executed. For every execution of the *"ShpF"* state, states *"MInp"* and *"InpF"* will be executed I_1 times. After K executions of state *"ShpF"* (and thus KI_1 executions of the *"mInp"* and *"InpF"* states), *"InpF"* goes back to state *"MShp"* to multiply the previously generated Gaussian variates with one of the shaping filter scaling factors.

When using an elliptic filter as the shaping filter, only K cascaded biquads and two multipliers are required, as shown in Figure 5.15(a). In order to implement the ILPFs, P cascaded biquads and $2P$ multipliers are required as shown in Figure 5.15(b). In this case the CDFG of the shaping filter and the first Chebyshev Type II low-pass filter are shown in Figure 5.16.

To implement the datapath shown in Figure 5.14(b), the GVG block uses the Gaussian variate generator described in Chapter 3. The registers of the datapath and the $K + P_1$ scaling factors of biquads are implemented using configurable slices while $4(K + P_1)$ registers and $4(K + P_1)$ coefficients of $K + P_1$ sos's are implemented using four dual-port BlockRAMs as shown in Figure 5.13(b). The datapath in Figure 5.14(b) utilizes 3% of the configurable slices, 1% of the dedicated multipliers and six on-chip BlockRAMs. The second ILPF was designed using a fourth-order low-pass inverse Chebyshev IIR filter and was implemented using one shared biquad, which receives its inputs either from the outputs of the first ILPF (i.e., the *"outI"* and *"outQ"* registers) or from "0" values inserted for zero padding. A complete fading channel simulator utilizes 4% of the configurable slices, 20% of the dedicated 18×18-bit multipliers, and 10 BlockRAMs, and operates at 50 MHz. An

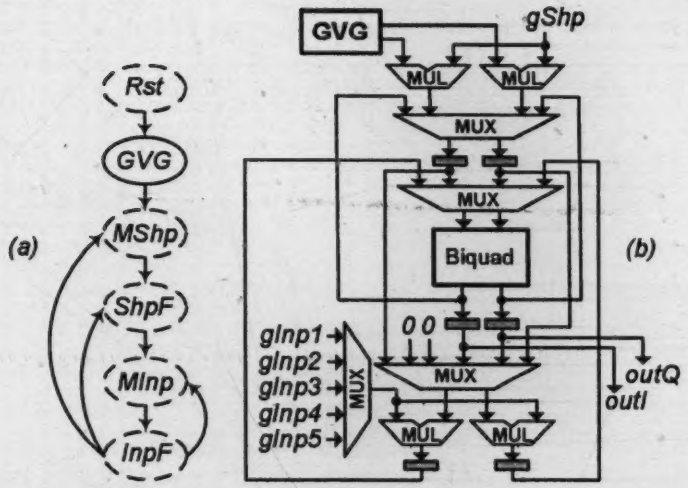

Figure 5.16: Control data flow graph of shaping filter and first-stage interpolation.

important property of the proposed scheme is that the complexity of the filter implementation will only impact the rate of fading variate generation and has almost no effect on hardware resource utilization. Even though the shaping and the low-pass interpolator filters are designed to emulate mobile channels sampled at 10 MHz, since the fading channel simulator runs at a high clock frequency of 50 MHz, for the example system, the simulator operates 1.25 times faster than the target sample rate. The simulator can then be slowed down to emulate a wide variety of different channel characteristics over bandwidths of up to 12.5 MHz.

To further speed up the fading sample generation rate, we utilized the commutative properties of computing the cascaded sos's and re-ordered the operations in such a way that the multiplication state *"MShp"* (*"MInp"*) can be performed simultaneously with the *"ShpF"* (*"InpF"*) state. Assume that R_i is the output register of the i-th multiplier and BR_i is the register at the output of the i-th biquad in the cascade structure of Figure 5.13(a). As given in Table 5.2, we utilized an out-of-order scheduling scheme to reduce the required number of clock cycles to execute N cascaded second-order sections from $2N$ in the sequential scheme down to $\lceil N/2 \rceil$. Since the last stage requires two clock cycles to generate one sample, a clock frequency of 20 MHz is required to generate 10 Msamples/sec. Since the designed fading channel simulator operates at 50 MHz, this approach provides 2.5 times speed up to generate fading variates up to 25 Msamples/sec. Increasing the sample generation throughput in this way requires $N = K + P_1 + P_2$ times the number of registers

Table 5.2: Out-of-order execution scheme for the cascade structure.

Clock no.	*State*	*Operations*
$6i+0$	1	$R_4 = \text{MUL}(g_4, BR_3);\ BR_1 = \text{Biq}(sos_1, R_1);$
$6i+1$	4	$R_3 = \text{MUL}(g_3, BR_2);\ BR_4 = \text{Biq}(sos_4, R_4);$
$6i+2$	3	$R_6 = \text{MUL}(g_6, BR_5);\ BR_3 = \text{Biq}(sos_3, R_3);$
$6i+3$	6	$R_5 = \text{MUL}(g_5, BR_4);\ BR_6 = \text{Biq}(sos_6, R_6);$
$6i+4$	5	$R_2 = \text{MUL}(g_2, BR_1);\ BR_5 = \text{Biq}(sos_5, R_5);$
$6i+5$	2	$R_1 = \text{MUL}(g_1, BR_0);\ BR_2 = \text{Biq}(sos_2, R_2);$

Table 5.3: Out-of-order scheduling and register renaming for the cascade structure.

Clock no.	*State*	*Operation*
$6i+0$	1	$R = \text{MUL}(g_4, BR_3);\ BR_1 = \text{Biq}(sos_1, R);$
$6i+1$	4	$R = \text{MUL}(g_3, BR_2);\ BR_4 = \text{Biq}(sos_4, R);$
$6i+2$	3	$R = \text{MUL}(g_6, BR_5);\ BR_3 = \text{Biq}(sos_3, R);$
$6i+3$	6	$R = \text{MUL}(g_5, BR_4);\ BR_6 = \text{Biq}(sos_6, R);$
$6i+4$	5	$R = \text{MUL}(g_2, BR_1);\ BR_5 = \text{Biq}(sos_5, R);$
$6i+5$	2	$R = \text{MUL}(g_1, BR_0);\ BR_2 = \text{Biq}(sos_2, R);$

compared to the sequential approach, as given in Table 5.2. An important point in Table 5.2 to note is that allocating a physical register for every instance can lead to an inefficient register allocation when a register can be re-used after its lifetime (when its present value is no longer needed). We consider the fact that R_i can be re-used after Biq_i reads its content, and BR_i can be re-used after multiplication by g_{i+1}, where g_{i+1} is the scaling factor of the Biq_{i+1}. The scheduling of operations after re-using the registers is given in Table 5.3, where only one register is utilized for the output of multiplier operations.

The HDL model of the proposed datapath was simulated to verify the accuracy of the results against the fixed-point software simulation results. A block of 50,000 complex Gaussian variate samples was generated and passed through the designed filters, then the statistical properties of the 2.5×10^7 generated complex fading variates were measured. Figure 5.17 plots the desired ACF along with the ACF of the samples generated with the new model. As Figure 5.17 shows, the generated ACF accurately matches the theoretical ACF. The LCR [23] of the envelope of the generated fading variates and the desired LCR are plotted in Figure 5.18. Here again a close match is observed. Finally, Figure 5.19 plots the PDF of the generated fading variates against the ideal PDF. These plots show that the new implemented fading channel simulator faithfully reproduces the properties of the reference model.

We also simulated a communication system with 4-PSK and 16-QAM modulations and zero-forcing equalization at the receiver to show the performance of the new fading simulator with fixed point arithmetic in Figure 5.20. Here, for each SNR, we transmitted

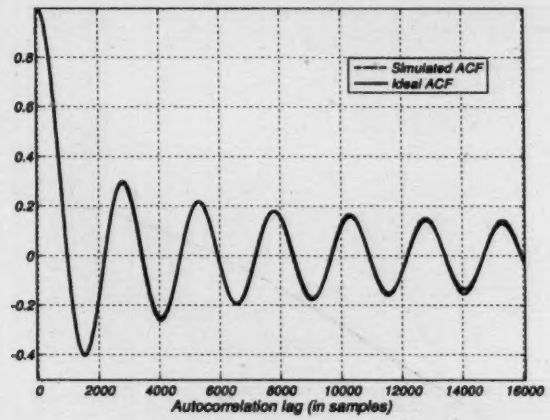

Figure 5.17: ACF and CCF of 2.5×10^7 generated fading variates.

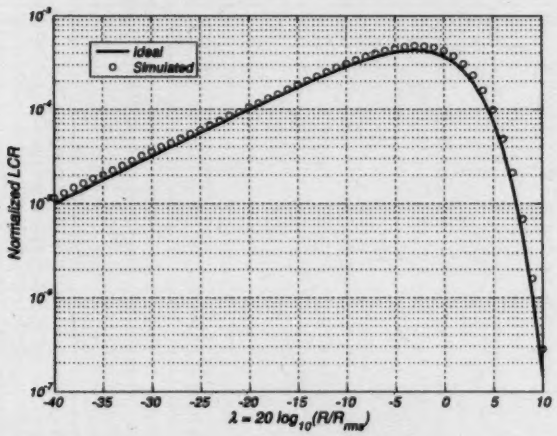

Figure 5.18: LCR of 2.5×10^7 generated fading variates.

10^{10} symbols and measured the average symbol error rate (SER). As Figure 5.20 illustrates, the SER plot generated by our new fading simulator again matches the expected theoretical target.

5.6 A Flexible Filter Processor for Fading Channel Emulation

The control structure for the cascaded shaping filter and ILPFs is rather straightforward; however, when simulating MIMO channels or frequency-selective channels, where there are multiple paths, each path possibly characterized with different filters with different specifications, a flexible implementation of control unit becomes more important. Rather than designing the control unit using random logic circuitry, we designed a flexible and compact

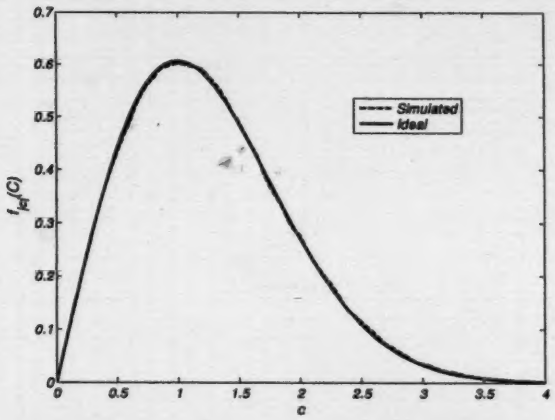

Figure 5.19: PDF of 2.5×10^7 generated fading variates.

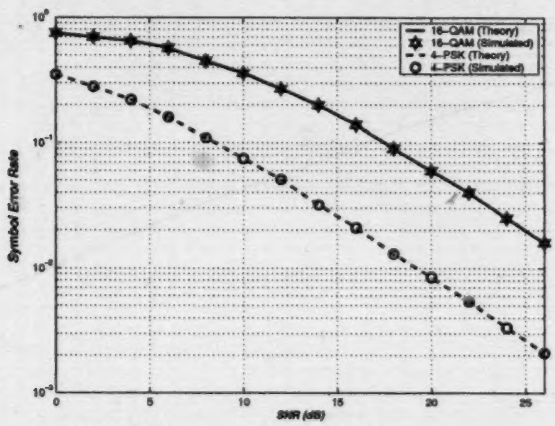

Figure 5.20: Symbol error rate for simulated 4-PSK and 16-QAM.

FP, called "Python", to efficiently implement the shaping filter and ILPFs on the FPGAs. The computation of V IIR filters (using N cascaded biquads) can be distributed among $m = 1, \cdots, M$ Python FPs.

Figure 5.21 shows the datapath of the Python FP. The core component is a biquad where its inputs coming from two memories (for the in-phase and quadrature parts), *"RAM RI"* and *"RAM RQ"*, with *"AD0"* as the read address bus and *"AD3"* as the write address bus. The outputs of the biquad are stored in two other memories *"RAM BI"* and *"RAM BQ"*, with *"AD1"* as the read address bus and *"AD3"* as the write address bus. ROM *"ROM g"* is initialized with the scaling factors of IIR filters with the *"AD2"* addressing bus. Two combinational multipliers are used to perform the scaling operation of intermediate values

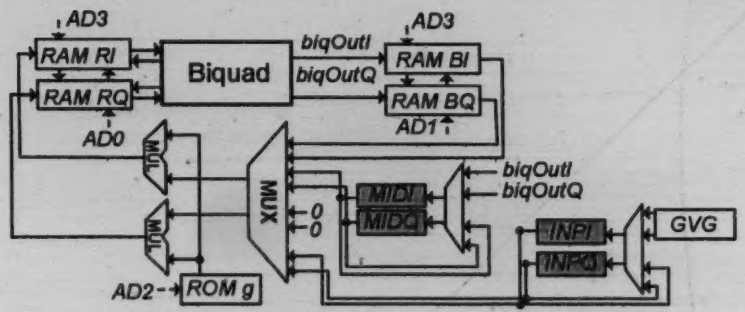

Figure 5.21: The architecture of the Python FP.

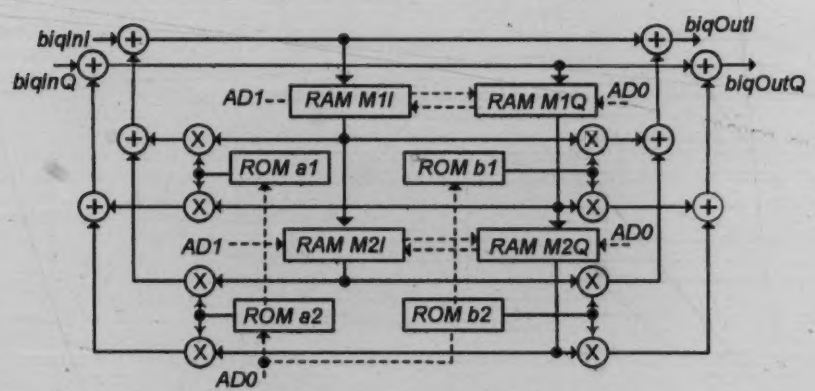

Figure 5.22: The datapath of biquad.

between biquads. The GVG generates two independent Gaussian variates. The zero inputs to the 4-input multiplexers are used when Python FP performs the zero padding operation required in interpolators.

The datapath of a biquad in Direct-Form II structure [147] is shown in Figure 5.22 where four intermediate variables are stored in four on-chip dual-port memories *"RAM M1I"*, *"RAM M1Q"*, *"RAM M2I"*, and *"RAM M2Q"*. Four coefficients are stored in four read-only memories (ROMs), *"ROM a1"*, *"ROM b1"*, *"ROM a2"*, and *"ROM b2"*. The *"AD0"* is the read address and *"AD1"* is the write address for the memories.

The Python controller is microprogrammable to ensure flexibility in the control unit. A code generator, written in *C* programming language, was developed that receives the specification of the shaping filter and the ILPFs as inputs and generates a sequence of microinstructions (microcode) to be executed by Python architecture. This microprogram is stored in an instruction ROM and is addressed by a program counter (PC). To minimize

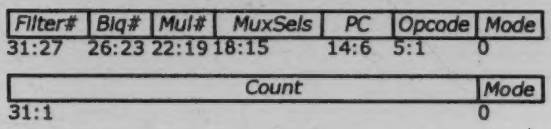

Figure 5.23: The microinstruction format.

the resource usage of hardware, we eliminated the random logic of the instruction decoder by utilizing horizontal microcode in which every control bit in microinstruction drives a control line in the Python datapath. The 32-bit microinstruction format is given in Figure 5.23. If *"Mode=1"*, the FP counts from the value given in the *"Count"* field down to zero (the wait operation); otherwise, it will execute the operation given by the 5-bit *"Opcode"* field. Hence, up to 32 different micro-operations (such as *BIQ*: execute a biquad operation, *MUL*: execute a multipler operation, *JMP*: jump to an address given by the PC, *RST*: reset the intermediate registers, etc.) can be defined. The *"PC"* field denotes the 9-bit program counter value. Four bits are used in the *"MuxSels"* field as multiplexer select lines. The *"Mul#"* *"Biq#"*, and *"Filter#"* fields denotes the multiplier number, the biquad number, and the filter number, respectively. The *"Mul#"* essentially is used since we utilized the commutative properties when computing the cascaded second-order sections. The second-order section operations can be re-ordered in such a way that the multiplication can be performed simultaneously with the biquad operation. This out-of-order scheduling scheme reduces the required number of clock cycles to execute N cascaded second-order sections from $2N$ in the sequential scheme down to $\lceil N/2 \rceil$.

One important design decision is how to bind operations of structure in Figure 5.13(a) onto one or more Python FPs. Assuming that the memories in Python architecture are addressed using AL bits, then up to 2^{AL} different biquad operations can be performed using one time-multiplexed biquad. For example, if $WL = 9$, then Python can execute 16 IIR filters each designed with up to 32 cascaded biquads. The maximum number of filters that can be executed with one FP depends on the size of memory, the number of biquads associated with each filter, and the minimum required throughput. Based on the given microinstruction format, 32 IIR filters, each designed with up to 16 biquads can be executed with one Python FP. But it is straightforward to modify the microinstruction format to support execution of various number of IIR filters, designed with different number of biquads (for example, 16 filters each designed with up to 32 biquads). Due to the slow variation of the samples in the shaping filter compared to the high operating rate of biquads implemented on FPGAs,

we bind the operation of the shaping filter and the first ILPF to be performed onto one Python FP. The second ILPF is implemented using a second Python FP to achieve the target sampling rate.

Assume that the shaping filter and the first stage ILPF is designed using K biquads and P_1 biquads, respectively, and is bound into one Python FP. After execution of every section of the shaping filter, the P_1 biquads of ILPF will be executed I_1 times where ILPF receives its input from the output of shaping filter once and $I_1 - 1$ times from zero input. Similarly, after KI_1 biquad executions of the shaping filter and the first-stage ILPF, the second stage interpolator will be executed I_2 times where P_2 biquads of the second-stage ILPF receives its input from the output of the first-stage ILPF once and $I_2 - 1$ times from zero.

The biquad datapath utilizes 1% of the configurable slices, 7% of the dedicated multipliers and eight on-chip BlockRAMs, and operates at 63.4 MHz. The Python datapath in Figure 5.21 utilizes 2% of the configurable slices, 9% of dedicated multipliers and 14 on-chip BlockRAMs. The second ILPF was designed using a fourth-order low-pass Chebyshev Type II IIR filter and was implemented using one Python processor. A complete fading channel simulator utilizes 4% of configurable slices, 20% of dedicated 18×18-bit multipliers, and 10 BlockRAMs, operates at 50 MHz. Even though the shaping and the low-pass interpolator filters are designed to emulate mobile channels samples at 10 MHz, since the fading channel simulator runs at a high clock frequency of 50 MHz, for the system at hand, the simulator operates 1.25 times faster than the target sample rate. The simulator can be slowed down to emulate a wide variety of different channel characteristics over bandwidths of up to 12.5 MHz.

The HDL model of the proposed datapath was simulated to verify the accuracy of the results against the fixed-point software simulation results. A block of $60,000$ complex Gaussian variate samples was generated and passed through the designed filters. Then the statistical properties of 3×10^7 generated complex fading variates were measured. Figure 5.24 plots the ideal ACF along with the ACF of the samples generated with the new model. As Figure 5.24 shows, the generated ACF accurately matches the ideal ACF. The LCR of the envelope of the generated fading variates and the ideal LCR are plotted in Figure 5.25. Here again a close match between the generated LCR and the ideal LCR can be observed. Also, Figures 5.26 plots the PDF of the generated fading variates against the ideal PDF. These plots show that the new implemented fading channel emulator faithfully reproduces the properties of the reference model.

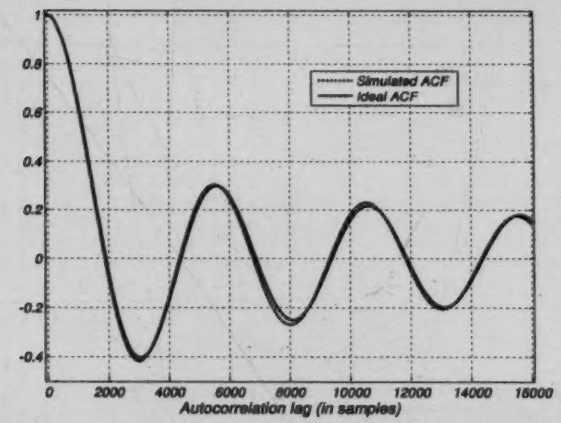

Figure 5.24: ACF and CCF of 3×10^7 generated fading variates.

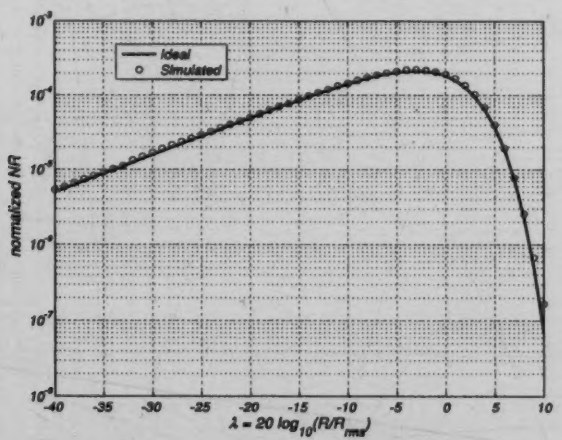

Figure 5.25: LCR of 3×10^7 generated fading variates.

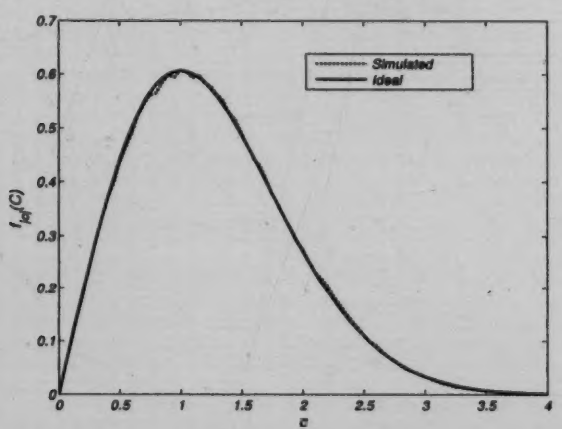

Figure 5.26: PDF of 3×10^7 generated fading variates.

5.7 Conclusions

Even though the filter-based fading simulator shows better statistical properties over the Jakes fading simulator, it can be more challenging to implement in hardware. In fact, the tight resource constraints imposed by contemporary FPGAs makes implementation of the computationally-intensive fading channel simulator a challenging task for designers. A novel computationally-efficient design and implementation scheme for fading channel simulators was presented. The filters required for shaping the power spectrum of the fading variates were designed to provide accurate fading samples and yet maintain minimum hardware complexity. Our fixed-point implementation of a Rayleigh fading channel simulator on a FPGA utilizes only 4% of the configurable slices, 20% of dedicated multipliers, and 10 (2%) BlockRAMs, while generating 25 million statistically accurate fading variates per second. Also, a flexible and compact filter processor architecture, called "Python", was designed to efficiently implement a multipath fading channel simulator on the FPGAs. Python uses a simple and short instruction set to generate multiple sequences of fading variates for simulating wideband and MIMO channels.

Chapter 6

An Efficient Parallel Architecture for LST Decoding in MIMO Systems

Spatial processing remains as the most promising, if not the last frontier, in the evolution of multiple access systems (A. Viterbi).

MIMO systems have emerged as an attractive new paradigm for spectrum-efficient wireless communications in rich multipath fading environments [2]. Figure 6.1 illustrates the model of a MIMO channel between $n_T > 1$ transmitter antennas and $n_R > 1$ receiver antennas, which collectively will be called an (n_T, n_R) *MIMO system*. The MIMO architecture can exploit diversity in both space and time to significantly increase system capacity as well as improve the quality (i.e., reduce the SER) of the wireless link in the presence of adverse propagation conditions, such as multipath fading and interference. Due to the high aggregate link capacity, an important first challenge is to minimize the computational complexity

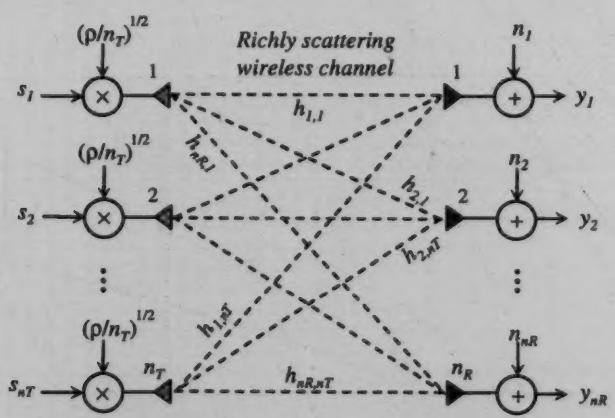

Figure 6.1: An (n_T, n_R) MIMO channel.

of the decoding algorithm at the MIMO receiver. Among the published non-linear decoding techniques [2], the *layered space-time* algorithms [154–157] employ a divide-and-conquer approach where, rather than jointly decoding the received symbols from all transmitter antennas, the receiver sequentially decodes one symbol at a time beginning, preferably, with the symbol with the highest SNR. The LST decoding algorithms then predicts and then subtracts away the interference due to the most recently decoded symbol from the simultaneously received signals, and then proceeds to decode the next symbol, and so on. The computational complexity of LST decoding is reported to be $\mathcal{O}(n^4)$ in the number n of antennas for the zero-forcing (ZF) and minimum mean-squared error (MMSE) LST algorithms [155], and $\mathcal{O}(n^3)$ for the square-root [158] and ordered QR-LST algorithms [159].

The great computational demands of MIMO signal decoding can exceed the performance available from even high-end DSPs. Therefore a second practical challenge in MIMO systems is to develop low-power, but sufficiently high-performance, hardware to implement the receiver. ASICs have been proposed to implement MIMO decoding algorithms [160, 161] and contemporary FPGAs have been used successfully to prototype MIMO testbeds [162–164]. The running time of an algorithm depends on the number of hardware instructions that need to be executed, and this number depends on the architecture of the processor. In the published LST decoding algorithms, there is abundant inherent parallelism that has yet to be exploited to increase the symbol decoding throughput at the receiver. Given a scalable parallel processor architecture, in particular, a key factor is the degree to which the algorithm can be parallelized and mapped efficiently onto the available processor resources. Therefore an alternative approach to using a faster conventional processor or an ASIC is to identify and exploit opportunities for parallel processing using a flexible parallel architecture to maximize the useful work that is accomplished in every clock cycle.

DSP-RAM is a moderately-parallel, scalable SIMD co-processor architecture for high-performance signal processing that was developed at the University of Alberta [165, 166]. In the *processor-in-memory* architecture [167] of DSP-RAM, a linearly-interconnected array of simple PEs is integrated with associated local memories. Integrating the processors with the memories exposes the enormous data bandwidth between the two, and eliminates the bottleneck that otherwise occurs on an external bus between the memory chips and processor(s) in conventional architectures. The degree of parallelism (i.e., the number of PEs) provides a trade-off between including more transistors on a die to increase the throughput

of a parallel algorithm, and running at a slower clock frequency to simplify the implementation, reduce the dynamic power consumption and still meet the required processing performance. In addition, reductions in the core operating voltage might be possible. Since dynamic power consumption is proportional to the voltage squared [168], the possibility of reducing the core voltage is especially attractive in power-constrained systems. We will show how DSP-RAM can be used to implement an LST MIMO receiver that offers high performance with relatively low power consumption. For a typical indoor wireless environment, a 100-MHz DSP-RAM can potentially provide more than 10 times greater decoding throughput at the receiver of a (4,4) MIMO system compared to a conventional 720-MHz DSP. The DSP-RAM processor has been coded in a HDL and synthesized for both readily available FPGAs and for a $0.18-\mu m$ CMOS standard cell implementation.

The rest of this chapter is organized as follows. The capacity of MIMO channels and the mathematical representation of MIMO systems are presented in Section 6.1. Section 6.3 presents different detection schemes for MIMO systems such as maximum likelihood, lattice decoders, and LST decoders. The related decoding algorithms and the key characteristics that make them suitable for parallel realization are discussed. The parallel DSP-RAM architecture is presented in Section 6.4. Section 6.5 discusses the mapping of the LST algorithm onto DSP-RAM. Section 6.6 describes the implementation of a MIMO receiver onto six different commercially available processors and an efficient realization of LST decoding onto the parallel DSP-RAM architecture. Finally, Section 6.7 makes some concluding remarks.

6.1 Mathematical Model of MIMO Systems

Let $h(\tau, t)$ be the time-varying channel response at time t to an impulse at time $t - \tau$. If signal $s(t)$ is transmitted, then the received signal $y(t)$ is given by

$$y(t) = h(\tau, t) * s(t) + n(t) = \int_0^{\upsilon} h(\tau, t) s(t - \tau) \mathrm{d}\tau + n(t)$$

where $h(\tau, t)$ is the complex envelope of the bandpass impulse response function, $n(t)$ is an AWGN signal, and υ is the duration of the causal channel impulse response. We assume that $s(t)$, $y(t)$, and $n(t)$ are modeled as the complex envelope of the underlying passband channel. Without loss of generality, some simplifying assumptions can be made to represent the discrete time (or sampled) baseband input/output model of SISO digital communication systems: (1) The channel bandwidth is 1 Hz and the symbol period T_s is 1 second. Hence, the average energy at the transmitter per symbol period E_s is equal to the total average transmit power P_T. (2) Since the transmission bandwidth is assumed 1 Hz, the noise power in the band is the same as power spectral density N_o. Therefore, noise power or noise PSD can be used interchangeably and can both be denoted by N_o. (3) The noise is assumed to be temporally-white *zero-mean circularly symmetric complex Gaussian* (ZMCSCG) with variance N_o. We denote a real Gaussian random variable with mean m_n and variance N_o as $n \sim \mathcal{N}(m_n, N_o)$, and a ZMCSCG random variable $n = n_i + j\, n_q$ as $n \sim \mathcal{CN}(0, N_o)$, where n_i and n_q are real i.i.d from $\mathcal{N}(0, N_o/2)$. For the special case where the real and imaginary part of noise components have $N_o = 1/2$, the complex noise has unit power. We assume that the noise samples $n[k]$ are i.i.d, i.e., $\mathrm{E}\{n[i]n^*[j]\} = N_o\delta[i - j]$. (4) Data symbols (prior to coding) are i.i.d and are drawn from scalar constellation $\mathbb{Q}$ with zero mean and unit average energy, i.e., unit average power when $T_s = 1$. (6) The received signal $y(t)$ is required to be oversampled, so we take multiple samples per symbol period T_s.

Assume that a sequence of complex symbols $s[\ell]$ ($\ell = 0, 1, 2, \cdots$) is to be transmitted over a SISO communication system. The received signal $y(t)$ can be written as [2]

$$\begin{aligned} y(t) &= h(\tau) * \Big[\sum_{\ell} \sqrt{E_s}\, s[\ell]\, \delta(t - \ell T_s) \Big] + n(t) \\ &= \sum_{\ell} \sqrt{E_s}\, s[\ell]\, h(t - \ell T_s) + n(t) \end{aligned}$$

where $h(\tau)$ denotes the continuous time baseband channel impulse response (the t dependence is omitted for clarity). If $y(t)$ is sampled at $t = kT_s$ ($k = 0, 1, 2, \cdots$), then the

sampled signal response is

$$y[kT_s] = \sum_{\ell} \sqrt{E_s}\, s[\ell]\, h[(k-\ell)T_s] + n[kT_s].$$

The signal response can also be written as

$$y[k] = \sum_{\ell} \sqrt{E_s}\, s[\ell]\, h[k-\ell] + n[k]$$

where $h[\ell]$, $\ell = 0, 1, 2, \cdots, L-1$, is the T_s-spaced sampled channel and L is the channel length measured in sampling periods. For a frequency-flat channel $h[i] = 0$ for $i \neq 0$. Denoting $h[\cdot]$ by the scalar channel gain h, the input/output relation for the channel simplifies to

$$y[k] = \sqrt{E_s}\, h\, s[k] + n[k] \tag{6.1}$$

where the principal impairments are *multiplicative* fading and *additive* noise. For the frequency-selective case, the received signal sampled at time index k is

$$y[k] = \sqrt{E_s} \Big[h[L-1], \cdots, h[1], h[0] \Big] \Big[s[k-L+1], \cdots, s[k-1], s[k] \Big]^T + n[k]$$

The mathematical representation of SISO channels can be extended to describe the MIMO system. Let $h_{i,j}(\tau, t)$ denotes the time-varying channel impulse response (also called the *multiplicative channel gain*) between the j-th transmitter and i-th receiver. The vector $\mathbf{h}_j = [h_{1,j}(\tau,t), h_{2,j}(\tau,t), \cdots, h_{n_R,j}(\tau,t)]^T$ is the channel induced by the j-th transmit antenna across the receive antenna array. In matrix notation the input/output relation of the channel may be written as $\mathbf{y} = \mathbf{H}(\tau, t) * \mathbf{s}(t) + \mathbf{n}(t)$ or, equivalently,

$$\mathbf{y}[k] = \sqrt{\frac{E_s}{n_T}}\, \mathbf{H}\, \mathbf{s}[k] + \mathbf{n}[k] \tag{6.2}$$

where $\mathbf{y}[k]$ is the $n_R \times 1$ received signal vector over the k-th symbol period, $\mathbf{s}[k] = (s_1[k], s_2[k], \ldots, s_{n_T}[k])^T$ denotes the vector of transmitted symbols (collectively called a *space-time* (ST) symbol) drawn from a finite complex signal constellation $\mathbb{Q}$, and $\mathbf{n}[k]$ is the spatio-temporally white ZMCSCG $n_R \times 1$ noise vector with variance N_o in each dimension.

In the case of frequency-selective fading, the channel matrix can be represented as $\mathbf{H}[\ell]$ where $\ell = 0, 1, \cdots, L-1$ and L is the maximum channel length of all component $n_R\, n_T$ SISO links. The received signal vector at time instant k may be expressed as

$$\mathbf{y}[k] = \sqrt{\frac{E_s}{n_T}}\, \mathbf{H} \Big[\mathbf{s}_1[k], \cdots, \mathbf{s}_{n_T}[k] \Big]^T + \mathbf{n}[k]$$

where each entry $\mathbf{h}_{i,j}$ can be written as $\mathbf{h}_{i,j} = \left[h_{i,j}[L-1], \cdots, h_{i,j}[0]\right]$ and $\mathbf{s}_j[k] = \left[s_j[k-L+1], \cdots, s_j[k-1], s_j[k]\right]^T$. For the case of flat-fading channels, the tap gains are assumed to be constant over the time period considered. The output at any instant of time is independent of the inputs at previous times, thus we can drop the time index k in (6.2) (and similarly in (6.1)) and express the input/output relation as

$$\mathbf{y} = \sqrt{\frac{E_s}{n_T}}\,\mathbf{H}\,\mathbf{s} + \mathbf{n}. \tag{6.3}$$

A few assumptions are usually made for analyzing MIMO systems: (1) If the mean energy per transmitted symbol is $E_{sj} = \mathrm{E}\{s_j^* s_j\}$, the total energy per use of the MIMO channel (i.e., simultaneous transmission of a ST symbol from n_T antennas) is $E_s = \sum_{j=1}^{n_T} E_{sj}$. E_s equals the total average transmit power P_T when the symbol period is unity. For example, if the transmitted symbols are drawn from a *unit average energy constellation* (i.e., $\mathrm{E}\{|\mathrm{s}_j|^2\} = 1$) so that each antenna transmits unity power, then $P_T = n_T$. The power constraint on the transmitted signal (independent of n_T) can be expressed as $\mathrm{E}\{\mathbf{s}^H\mathbf{s}\} \leq P_T$ (or the covariance matrix $\mathbf{C_{ss}} = \mathrm{E}\{\mathbf{s}\mathbf{s}^H\} \leq P_T/n_T\mathbf{I}_{n_T}$) [2]. The symbol energy (i.e., the power launched by each transmitter) can be reduced by n_T as E_s/n_T, so that the total radiated power is constant and independent of n_T for a fixed SNR. (2) The AWGN vector $\mathbf{n}$ has equal variance components $n \sim \mathcal{CN}(0, N_o)$, where N_o is the noise power (variance) on each receiving antenna. For the ZMCSCG noise vector $\mathbf{n}$, the covariance matrix can be written as $\mathrm{E}\{\mathbf{n}\mathbf{n}^H\} = N_o\,\mathbf{I}_{n_R}$ [2]. The noise at the receiver is assumed to be independent of both the data and the channel. (3) The SNR per receiver antenna ρ can be defined as the ratio of the total transmitted power per channel used (P_T) divided by the per-component noise variance (N_o). Therefore, if the n_R noise components of $\mathbf{n}$ are assumed to be i.i.d. $\mathcal{CN}(0,1)$, then the average transmitted power is equal to the average SNR. (4) For richly scattered propagation, the $h_{i,j}$ are usually modeled as i.i.d, ZMCSCG random variables (i.e., $\mathrm{E}\{h_{i,j}\} = 0$ and $\mathrm{E}\{h_{i,j}h_{m,n}^*\} = 0$ if $i \neq m, j \neq n$) with equal unit variance (i.e., $\mathrm{E}\{|h_{i,j}|^2\} = 1$) [2]. With $h_{i,j}$ being complex Gaussian and uniformly distributed in phase, the magnitude $|h_{i,j}|$ is Rayleigh-distributed. Also, the column vectors of complex transfer gains from the j-th transmitter antenna to all n_R receiver antennas $\mathbf{h}_j$ are assumed to be independent, which corresponds to the diversity provided in richly-scattered environments. Uncorrelated scattering is usually ensured by physically separating the antennas at the transmitter and receiver by a few carrier wavelengths [169].

6.2 Algorithms for Systems of Linear Equations

As given in Equation (6.3), the signal at each receiver antenna is the superposition of transmitted symbols scaled by the channel gain and corrupted by AWGN. The conventional signal processing algorithm at the receiver consists of two main steps:

(I) Estimate the channel matrix through a training phase. We make the common assumption that communication is carried out in bursts of data alternating with training signals. The quasi-stationary viewpoint assumes that the channel conditions are fixed during a burst. This is a reasonable assumption for a communication system where the burst duration is much less than the channel coherence time. For each burst of received information, the "fixed" channel response can be estimated using different estimation schemes [42]. To account for changing conditions, the channel is often assumed to change randomly between bursts due to accumulated changes in the channel fading. The receiver will be assumed to have previously estimated the propagation matrix $\mathbf{H}$ using the training signals before commencing the decoding process for the following data. Note that $\mathbf{H}$ is assumed to be known to the receiver but not to the transmitter [170].

(II) The n_T components of a transmitted ST symbol $\mathbf{s}$ must be recovered from the received signal vector $\mathbf{y}$ and the previously-estimated channel matrix $\mathbf{H}$ [170].

Before discussing various decoding techniques for recovering the transmitted symbols $\mathbf{s}$ using the linear complex-valued system Equation (6.3), lets consider the linear system transformation

$$\mathbf{Hs} = \mathbf{y} \tag{6.4}$$

where $\mathbf{s}$ is a vector of $n_T \times 1 \in \mathbb{R}^{n_T}$ unknown values, $\mathbf{H}$ is an $n_R \times n_T$ matrix of real values, and $\mathbf{y}$ is a vector of $n_R \times 1 \in \mathbb{R}^{n_T}$. If $n_T = n_R$ (i.e., there are as many equations as unknowns), then there is a chance to obtain a unique solution for $\mathbf{s}$. If one or more of n_R linear equations is a linear combination of the others, then there is no unique solution and the system is called *singular*. If $n_R < n_T$, then there are fewer equations than unknowns. In this case, either there is no solution or there is more than one solution for $\mathbf{s}$ [171]. When $n_R \geq n_T$, various techniques can be used to solve system of equations in (6.4) [172]. Each technique has a particular computational complexity and error. By error we mean a norm $\|\mathbf{s} - \hat{\mathbf{s}}\|$ of the difference between the true solution $\mathbf{s}$ and an approximation $\hat{\mathbf{s}}$. While the precise choice of norm does not affect the cost significantly [173], we can think of $\|\cdot\|$ as

a matrix norm. The *absolute error* $\|\mathbf{s}-\hat{\mathbf{s}}\|$ can be defined as

$$\|\mathbf{s}-\hat{\mathbf{s}}\| = \|\mathbf{H}^{-1}\mathbf{H}(\mathbf{s}-\hat{\mathbf{s}})\| = \|\mathbf{H}^{-1}(\mathbf{y}-\mathbf{H}\hat{\mathbf{s}})\| \leq \|\mathbf{H}^{-1}\|.\|\mathbf{y}-\mathbf{H}\hat{\mathbf{s}}\| \tag{6.5}$$

where $\mathbf{y}-\mathbf{H}\hat{\mathbf{s}}$ is called the *residual*. The *relative error* in $\hat{\mathbf{s}}$ can be defined as

$$\frac{\|\mathbf{s}-\hat{\mathbf{s}}\|}{\|\hat{\mathbf{s}}\|} \leq \|\mathbf{H}^{-1}\|.\|\mathbf{H}\|.\frac{\|\mathbf{y}-\mathbf{H}\hat{\mathbf{s}}\|}{\|\mathbf{H}\|.\|\hat{\mathbf{s}}\|} \tag{6.6}$$

where $\kappa(\mathbf{H}) = \|\mathbf{H}^{-1}\|.\|\mathbf{H}\|$ is called the *condition number* of $\mathbf{H}$. Clearly, the *error bound* is proportional to $\|\mathbf{H}^{-1}\|$. Under the block-fading assumption of wireless channels, we are interested in computing the inverse of estimated channel $\mathbf{H}^{-1}$ once and then using it for decoding a relatively long block of received symbols.

Due to the high computation complexity of matrix inversion and the fact that the matrix inversion has to be calculated for each block of transmitted ST symbols, rapid channel inversion is challenging to implement in practice. Two major schemes have been proposed to perform the matrix inversion operation: the *direct routines* (i.e., routines that execute in a predictable number of operations, typically of the order of $\mathcal{O}(n^3)$) and the *iterative methods* (i.e., methods that do not obtain an exact solution in finite time, but that attempt to converge to a solution asymptotically). A numerically stable deterministic technique for computing the pseudo-inverse is to use singular value decomposition (SVD) [158, 174]. The complexity of performing the SVD of an $n_R \times n_T$ matrix $\mathbf{H}$, is $2n_R^2 n_T + 11n_T^3$ and the complexity of finding the pseudo-inverse $\mathbf{G} = \mathbf{H}^{\dagger} = (\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H$ is $2n_R n_T^2$ [158]. The *Gauss-Jordan* (GJ) and *Gauss-Elimination* (GE) techniques are two other deterministic approaches that require the right-hand side of the Equation (6.4) [86]. *LU Decomposition* (LUD) does not share the previous deficiency, and also has a small operation count, both for solving (6.4), and also for matrix inversion. *QR factorization* decomposition is another deterministic technique which involves about twice as many operations as LU decomposition. The *Cholesky decomposition* of a symmetric and positive definite matrix can be performed up to twice faster than LU decomposition. But this is rather too specific to be used for general channel matrices. Iterative methods (such as Jacobi and Gauss-Seidel algorithms) often yield a solution within acceptable precision after a small number of iterations and, therefore, become preferable for large matrices or when the problem is close to singular (i.e., the system is not linearly dependent but the round-off error could make the system singular) [86]. Iterative methods may also have smaller storage requirements than direct methods [44].

If linear transformation in (6.4) is a complex-valued system, then we may re-write it

using quadrature components as

$$(\mathbf{H}_i + j\mathbf{H}_q)(\mathbf{s}_i + j\mathbf{s}_q) = \mathbf{y}_i + j\mathbf{y}_q \tag{6.7}$$

or in vector notation as

$$\begin{pmatrix} \mathbf{H}_i & -\mathbf{H}_q \\ \mathbf{H}_q & \mathbf{H}_i \end{pmatrix} \cdot \begin{pmatrix} \mathbf{s}_i \\ \mathbf{s}_q \end{pmatrix} = \begin{pmatrix} \mathbf{y}_i \\ \mathbf{y}_q \end{pmatrix}. \tag{6.8}$$

Assume that $n_R = n_T = n$. One may solve the $n \times n$ complex system in (6.7) using complex arithmetic or solve the $2n \times 2n$ real system in (6.8). While the second approach is inefficient in storage (since $\mathbf{H}_i$ and $\mathbf{H}_q$ are stored twice), it is also shown in [175] that complex matrix inversion can be up to twice as fast as real matrix inversion for $n \geq 3$. Moreover, the rounding error bound of the complex computation is less than that of the real system.

If the multipath scattering is sufficiently rich, the transmitted signals are scattered slightly differently since they originate from different transmitter antennas and propagate over different paths. Consequently, if the MIMO system equations in (6.3) are sufficiently independent, a decoding algorithm at the receiver can recover the symbols despite the multitude of interferers. When there is no noise, i.e., $N_o = 0$, the exact solution of Equation (6.3) can be found in $\mathcal{O}(n^3)$ scalar operations, using a matrix pseudo-inverse operation. When N_o approaches infinity, the received vector $\mathbf{y}$ becomes increasingly random and the exact solution has *exponential* complexity in n [42]. For intermediate noise levels, tractable algorithms (i.e., polynomial time algorithms) are required to achieve an acceptable error rate with realizable computational-complexity.

6.3 MIMO Decoding Techniques

Several techniques have been proposed for recovering the symbols transmitted by n_T antennas [3, 42, 154–159, 176–178]. From estimation theory, the optimal decoding method with respect to the error rate is *maximum likelihood* (ML), where the receiver considers all possible ST symbols that could have been transmitted. For a MIMO system that transmit the uncoded data stream from n_T antennas through a frequency-flat MIMO channel, the ML receiver chooses symbol $\hat{\mathbf{s}}$ as

$$\hat{\mathbf{s}} = \arg \min_{\mathbf{s} \in \mathbb{Q}^{n_T}} \left\| \mathbf{y} - \sqrt{\frac{E_s}{n_T}} \mathbf{H}\,\mathbf{s} \right\|_2^2 \tag{6.9}$$

where the minimization is performed over all possible n_T-element vectors $\mathbf{s} \in \mathbb{Q}^{n_T}$. The ML detection problem can be posed as an integer least-squares problem that can be solved via many different algorithms [172]. Because ML decoding requires an exhaustive search over a typically large set $\mathbb{Q}^{n_T}$, its computational complexity can be high, and probably prohibitive when many antennas and/or high-order modulations are used [179].

To gain some perspective on the relative complexity of ML decoding, consider an (n_T, n_R) system that transmits m b/s of data using q-QAM over a channel within k Hz bandwidth. If we assume that the data occupies most of the available time in a burst and ignore the training signals, channel characteristics and any other in-band control data, then $v = \frac{m}{n_T \log_2 q}$ complex symbol vectors are received per second. To calculate Equation (6.9) over q^{n_T} possibly valid constellation points $\mathbf{s}$ for each received vector, one can verify that $q^{n_T}\big(n_R(n_T+1)\big)$ complex multiplications, $q^{n_T}\big(n_R(n_T+1)-1\big)$ complex additions and $q^{n_T}-1$ comparisons are required. As an example consider a $(4,4)$ system that operates at 19.2 Mb/Sec in a bandwidth of 1.8 MHz, and utilizes uncoded 16-QAM in each transmitter. Assume further that a DSP is present in each receiver that can perform one complex multiplication, addition or comparison operation in each clock cycle. To decode each received vector $\mathbf{y}$ using a single purely-sequential processor [41], 2,623,779 clock cycles will be required. To decode the 1.2×10^6 complex symbol vectors per second, the DSP must operate at a 3148 GHz clock frequency, which is far beyond current processor technology. Current degrees of parallelism in contemporary DSPs do not provide enough speed-up to change this situation. Even though the complexity of ML decoding is often too great, the algorithm clearly has potentially exploitable parallelism when calculating $\|\mathbf{y} - \mathbf{H}\,\mathbf{s}\|$ over all possible $\mathbf{s} \in \mathbb{Q}^{n_T}$. If the algorithm could be parallelized and mapped onto a parallel computer architecture with n_P PEs, the throughput could be directly multiplied by n_P (in the optimal case).

To provide computational saving over ML decoding, the *universal lattice decoders* [180, 181], a class that includes sphere decoders, can be used to decode the received signals in MIMO systems [182, 183]. The main idea behind *sphere decoding* (SD) is to reduce computational complexity by searching over only those lattice points (defined as $\mathbf{Hs}$) that lie within a hypersphere of radius d_o around the received signal $\mathbf{y}$, rather than searching over the entire lattice (See Figure 6.2). The complexity of SD is dominated by the amount of processing required to search for the points inside the present hypersphere, the number of points in the initial hypersphere of radius d_o, the dimension $2n_T$ of the search space,

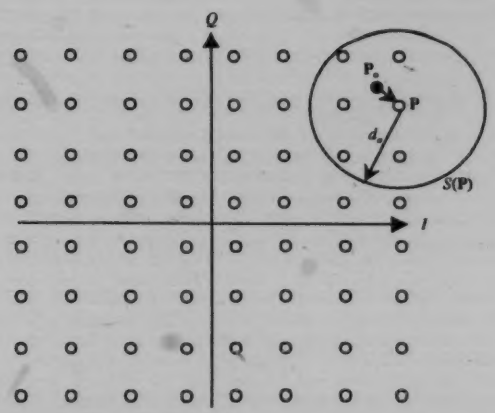

Figure 6.2: Sphere around the decoded point $\mathbf{p} = \mathcal{Q}(\mathbf{H}^{\dagger}\mathbf{y})$.

and the calculation required to determine the lattice point within the hypersphere closest to a preliminary decoded point [178]. An important property of the sphere decoding (SD) algorithm, as claimed in [181], is that its complexity is independent of the lattice constellation size, which makes SD attractive for high data rate transmission. In order to determine which of the lattice points lie inside the given sphere, Fincke and Pohst proposed a decoding algorithm in [184]. They showed that if b^{-1} is a lower bound on the eigenvalues of the Gram matrix $\mathbf{G} = \mathbf{H}^{H}\mathbf{H}$, then the required number of arithmetic operations (additions, subtractions and multiplications) is

$$\mathcal{O}\Big(n_R^2 \times (1 + \frac{n_R - 1}{4bd_o})^{4bd_o}\Big). \tag{6.10}$$

For a reasonable choice of the initial radius $d_o = b^{-1}$, the above complexity can be approximated by $\mathcal{O}(n_R^6)$. In the presence of deep fades, many points fall inside the search hypersphere and decoding can be very slow. In fact, the worst-case complexity of SD is exponential. This is evident from the fact that the Gram matrix may have a very small eigenvalue, which gives a relatively large exponent b in Equation (6.10) [181]. In [40] the complexity of SD is defined as the number of multiplications carried out until the closest point within the hypersphere is found. However, trying to build an exact expression for the number of arithmetic operations for SD is not useful since the required number of iterations is variable. In [42] the complexity of SD is considered with respect to the noise variance and the dimension of the lattice. It is shown that, over a wide range of noise variances and values of n_T, the expected complexity is polynomial, and in fact is often roughly cubic in n_T.

Even though the original SD algorithm significantly reduces the computational complexity of decoding compared to ML, many shortcomings and potential inefficiencies exist. First, in the presence of noise, for each received point a different hypersphere, with different bounds that depend on the noise variance (and eventually fading), must be calculated. Therefore, SD is inherently an irregular algorithm where the size of the hypersphere around the received point varies unpredictably. In particular, irregularity arises in the number of iterations required to calculate the hypersphere parameters and, consequently, in the number of points enclosed in each hypersphere. Hence the complexity of the algorithm will itself be a random variable [42]. Second, the hypersphere around the received point is constructed after first calculating minimum and maximum bounds for each vector component. To perform this calculation, a relatively complicated control sequence must be executed and this execution sequence is not deterministic (i.e. predictable) at compile time. Third, most contemporary high-performance processors support some form of instruction-level parallelism (superscalar or very long instruction word) [41], data-level parallelism (multiple functional units) and/or subword parallelism using a reconfigurable datapath. Mapping an algorithm that has an irregular and/or random control sequence, such as SD, onto a data-parallel processor significantly decreases the efficiency because of the increased number of pipeline stalls during execution. Stalls are caused by the *dependencies* between program instructions that reduce the actual speedup that can be achieved [43].

A more computationally-efficient approach is to use *heuristic* methods (i.e., may not always achieve the correct result, but usually produces an acceptable solution) in which a linear filter is typically used to separate the received signal into its component transmitted data streams and then decode each stream independently. The matrix filter

$$\mathbf{G}_{ZF} = \sqrt{\frac{n_T}{E_s}}\mathbf{H}^{\dagger} \tag{6.11}$$

separates the received signal into its component transmitted symbols, where the superscript $\dagger$ denotes the Moore-Penrose pseudo-inverse operator and $\mathbf{H}^{\dagger} = (\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H$ with a computational complexity of $\mathcal{O}(n^3)$ [174]. $\mathbf{G}_{ZF}$ is an $n_T \times n_R$ inverted channel matrix, commonly called a *zero-forcing matrix*. The output of ZF receiver is given by $\mathbf{z} = \mathbf{s} + \mathbf{G}_{ZF}\mathbf{n}$, where we assume that $n_R \geq n_T$ and $\mathbf{H}$ has full column rank. As opposed to ZF linear receivers for SISO and SIMO channels, which were constructed to cancel ISI, here they are used to cancel multistream interference (MSI). The ZF receiver decomposes the MIMO link into n_T parallel streams, each with diversity gain proportional to $n_R - n_T + 1$

out of a maximum possible n_R diversity [2]. Even though the noise is enhanced by $\mathbf{G}_{ZF}$ and correlated across the channels, each received symbol is decoded independently ignoring noise correlation.

The ZF receiver eliminates MSI completely at the expense of noise enhancement. The MMSE receiver balances MSI mitigation with noise enhancement and minimizes the total error [15]. The $\mathbf{G}_{MMSE}$ can be written as [2]:

$$\mathbf{G}_{MMSE} = \sqrt{\frac{n_T}{E_s}}\left(\mathbf{H}^H\mathbf{H} + \frac{n_T}{\rho}\mathbf{I}_{n_T}\right)^{-1}\mathbf{H}^H. \tag{6.12}$$

At low SNR, the MMSE outperforms the ZF receiver that continues to enhance noise. At high SNR, the MMSE receiver converges to a ZF receiver as $\mathbf{G}_{MMSE} \simeq \mathbf{G}_{ZF}$. While the MMSE receiver is superior to the ZF receiver, it requires an accurate estimate of the value of SNR at the receiver.

Another heuristic approach is to use the iterative QR decoding scheme. The $n_R \times n_T$ channel matrix $\mathbf{H}$ is decomposed using Gram-Schmidt orthogonalization into an $n_R \times n_T$ unitary matrix $\mathbf{Q}$ and an $n_T \times n_T$ upper triangular matrix $\mathbf{R}$ with non-zero entries on the diagonal [172]. The columns of $\mathbf{Q}$ form an orthonormal basis, $\mathbf{q}_1, \ldots, \mathbf{q}_{n_T}$, and accordingly $\mathbf{Q}^H\mathbf{H} = \mathbf{R}$. By left multiplying Equation (6.3) with $\mathbf{Q}^H$, an estimate $\hat{\mathbf{y}}$ of the actual received vector $\mathbf{y}$ is created:

$$\mathbf{Q}^H\mathbf{y} = \hat{\mathbf{y}} = \mathbf{R}\mathbf{s} + \mathbf{Q}^H\mathbf{n} = \mathbf{R}\mathbf{s} + \hat{\mathbf{n}}$$

Since $\mathbf{Q}$ is unitary, the statistical properties of the noise terms $\hat{\mathbf{n}} = \mathbf{Q}^H\mathbf{n}$ and n are the same. Element k of vector $\hat{y}_k$ becomes

$$\hat{y}_k = r_{k,k}\, s_k + f_k + \hat{n}_k, \text{ where } f_k = \sum_{j=k+1}^{n_T} r_{k,j}\, s_j.$$

Note that the element $\hat{y}_k$ depends on the transmitted signal s_k, the interference term f_k, and the noise component $\hat{n}_k$. Since $\mathbf{R}$ is upper triangular, signals with larger indices avoid interference from signals with small indices. In other words, the interference nulling pattern is created directly by the unitary transformation where f_k is independent of $s_1, \ldots, s_{k-1}$. Thus the interference term f_k can be cancelled by using previous decisions $\hat{s}_{k+1}, \ldots, \hat{s}_{n_T}$ and therefore $\hat{y}_k - \sum_{j=k+1}^{n_T} r_{k,j}\hat{s}_j = r_{k,k}\, s_k + \hat{n}_k$. Symbol $\hat{s}_k$ can be estimated by

$$\hat{s}_k = Q\left(\frac{\hat{y}_k - \sum_{j=k+1}^{n_T} r_{k,j}\hat{s}_j - \hat{n}_k}{r_{k,k}}\right)$$

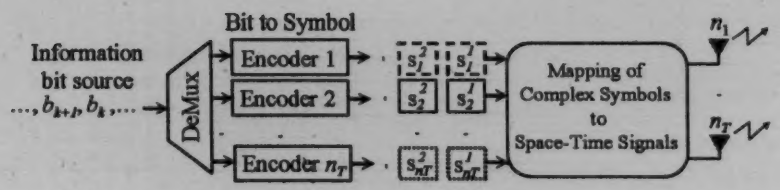

Figure 6.3: LST transmitter architecture.

where $\mathcal{Q}$ is the quantization function appropriate for the signal constellation $\mathbb{Q}$ in use.

The non-linear *Bell Labs layered space-time* (BLAST) algorithm is a divide-and-conquer decoding strategy based on successive interference cancellation [3, 155, 176] for a LST architecture [3, 179]. The LST MIMO architecture introduces and then exploits temporal and spatial diversity in the transmitted signals. A block diagram of a conventional single-user LST system is shown in Figure 6.3. The incoming information bits are denoted by $\{b^k\}$, where k is a discrete time index. The high-rate data stream is demultiplexed into n_T equal-capacity parallel substreams, called *layers*. Each layer is then encoded separately using the same constellation. If a block of information consists of L space-time symbols, then the output of the n_T encoders can be represented by the following $(n_T \times L)$ ST codeword matrix $\mathbf{S}$:

$$\mathbf{S} = \begin{bmatrix} s_1^1 & s_1^2 & \dots & s_1^L \\ \vdots & \vdots & \vdots & \vdots \\ s_{n_T}^1 & s_{n_T}^2 & \dots & s_{n_T}^L \end{bmatrix}. \tag{6.13}$$

Matrix $\mathbf{S}$ comprises the symbols that are transmitted by n_T transmitter antennas at L different time instants. The resulting ST signals drive identical transmitter pulse filters and the resulting digital baseband signals are modulated by a carrier and broadcast by n_T antennas. All transmitter antennas are assumed to use the same constellation and to transmit data simultaneously using the same carrier frequency and symbol timing in the same frequency band.

The LST receiver algorithm consists of two phases [155]: First, the channel matrix is estimated [185]. The channel state information (CSI) is assumed to be known to the receiver but not to the transmitter [170]. Second, the received data signals from one ST symbol interval are processed to recover the n_T transmitted complex symbols, $(s_1^j, \dots, s_{n_T}^j)$, within a fixed number of symbol times after time j. The decoding algorithm proceeds iteratively through the following three steps until all n_T symbols are recovered.

Step 1) Interference nulling: Interference nulling tries to reduce the amount of interfer-

ence towards s_k by multiplying the received signal $\mathbf{y}$ by a nulling vector $\mathbf{g}_k$ [156]. Consequently, the symbol rate processing has a computational complexity of only $\mathcal{O}(n^2)$. In ZF-LST, $\mathbf{g}_k$ can be calculated using the k-th row of nulling matrix $\mathbf{G} = \underline{\mathbf{H}}_k^{\dagger}$ where the notation $\underline{\mathbf{H}}_k$ denotes the matrix obtained by zeroing column k of $\mathbf{H}$. Since all components of a transmitted vector $\mathbf{s}$ are assumed to utilize the same constellation, the component s_j with the lowest post-detection SNR will dominate the error performance of the detection process. It was shown in [154] that starting with the symbol (layer) with the strongest post-detection SNR, and then proceeding successively to the symbol with the weakest SNR, improves the performance remarkably [155]. This corresponds to choosing the row $\mathbf{g}_k$ of $\mathbf{G} = \underline{\mathbf{H}}_k^{\dagger}$ with the minimum norm, $k = \min_i \|\mathbf{g}_i\|^2$, and selecting the corresponding row as the nulling vector in the interference nulling step. Thus, the k-th element of $\mathbf{s}$ with the highest SNR is detected by $\hat{s}_k = \mathbf{g}_k \mathbf{y}$.

Step 2) Symbol decoding: Symbol s_k from the k-th transmitter antenna is estimated by mapping to the nearest symbol $s_k = \mathcal{Q}(\hat{s}_k)$ in the constellation, where $\mathcal{Q}(\cdot)$ function calculates the Euclidean distance between $\hat{s}_k$ and the symbols in the constellation.

Step 3) Symbol cancellation: At this stage, the recovered symbol s_k can be used to improve the estimate of the remaining $n_T - 1$ symbols that are yet to be recovered. The interference on the $n_T - 1$ other signals due to s_k can be subtracted out from the received signal as $\mathbf{y}' = \mathbf{y} - s_k \mathbf{h}_k$. Thus, the number of signals remaining to be detected is reduced by one with each decoding step, while the number of receiver antennas stays the same. Therefore, the diversity level of the resulting system should increase going from layer to layer [186].

The above three steps are repeated to recover the $n_T - 1$ remaining symbols that were transmitted at the same symbol time.

Another way to improve the detection performance, especially for mid-range SNR values, is to replace the ZF nulling proposed in [3, 176] by the more powerful MMSE algorithm. In addition to nulling out the interferers, the noise level on the channel is taken into account. A disadvantage is, however, that the SNR has to be determined somehow at the receiver. MMSE nulling utilize the projection matrix $\mathbf{G}$ given in (6.12). The MMSE-LST algorithm is summarized in Algorithm 7 [176], where the inputs are the channel matrix $\mathbf{H}$, the projection matrix $\mathbf{G}$, and the received ST symbol $\mathbf{s}$. The output of the algorithm is the decoded ST symbol.

An improvement over the ZF-LST and MMSE-LST algorithms was proposed in [158,

Algorithm 7 MMSE-LST algorithm

$k = \min_j \|\mathbf{g}_j\|^2$; g_j is the j-th row of $\mathbf{G}$
for $(i = 1; i \leq n_T)$ **do**
  $\hat{s}_k = \mathbf{g}_k \mathbf{y}$;
  $s_k = \mathcal{Q}(\hat{s}_k)$;
  $\mathbf{y} = \mathbf{y} - \mathbf{h}_k s_k$;
  $\mathbf{G} = \underline{\mathbf{H}}_k^{\dagger}$;
  $k = \min_i \|\mathbf{g}_i\|^2$;
end for

177] and is called the *square root algorithm*. In this technique, the nulling process was performed by the use of unitary transformations to avoid repeatedly evaluating the pseudo-inverse of the deflated matrices. This approach reduces the complexity from $\frac{27}{4}n_T^4$ for MMSE nulling to $\frac{29}{3}n_T^3$ for $n = n_T = n_R$, *i.e.* $\mathcal{O}(n^3)$, and increases the numerical stability compared with the original BLAST algorithm. Even though the square root algorithm offers an order of magnitude reduction in the computational complexity compared to MMSE-LST decoding, it is shown in [157, 158] that for a typical number of antennas (e.g., between one to eight) the complexity of the algorithm in floating-point operations (FLOPs) is almost the same as for other LST decoding algorithms.

The required decoding rate and error rate performance are two important decoding algorithm metrics. However, the microarchitecture of the target processor can greatly influence the decoding algorithm running times, so one must be careful to choose the most appropriate decoding algorithms for different processors. For example, depending on the microarchitecture of the processor, the interconnect topology and number of PEs, the complexity of the parallelized matrix inversion can be varied. For instance, it can be performed in $\mathcal{O}(\log^2 n)$ time using n^4 PEs on a three-dimensional reconfigurable mesh [187].

In either of the above LST decoding algorithms, there is much data-level parallelism that we propose to exploit using a moderately-parallel architecture. Even though both the square-root and ordered QR algorithms provide better numerical stability and less computational complexity than the MMSE-LST algorithm [158, 159], they exhibit load imbalance, and hence a less efficient hardware utilization, when they are mapped to a linear parallel architecture [174]. Therefore, we used MMSE-LST as the default decoding technique. However, our parallel implementation, described below, could be modified to accommodate any of the above decoding algorithms.

Table 6.1: Comparison of PIM architectures.

	Terasys	*IMAP*	*VIRAM*	*C•RAM*
Memory Technology	SRAM	SRAM	DRAM	DRAM
Word Width	1-bit	8-bit	Programmable	1-bit
Operands	Memory	Registers	Registers	Memory
Integer Multiply	No	Partial[1]	Yes	No
Floating Point	No	No	Yes	No
Controller	External	External	Internal	External

6.4 A Parallel Architecture for Digital Signal Processing

For problems with substantial data-parallelism, the SIMD architecture is often well-suited to achieving high processing rates. In such cases, the data can be distributed into many different independent pieces, and multiple PEs can operate on them simultaneously. SIMD machines come in two major flavors: (1) *processor array architecture*, which consists of a relatively large number of simple PEs, and (2) *vector processors* that have only a small number, typically between 1 and 32, of deeply-pipelined powerful execution units. The majority of today's high-performance microprocessors and DSPs include SIMD instructions to boost their performance on data-parallel applications [188].

To decrease the processor-memory performance gap, the processor-in-memory organization combines processing elements into memory. Table 6.1 presents major characteristics of four examples of PIM-style SIMD processors. Terasys is a massively-parallel SIMD processing array comprising 32,768 bit-serial PEs [167]. Since bit-serial processors require many clock cycles to compute a single value for fixed-point numbers, they achieve great performance only through massive parallelism. Each PE has access to a 2-Kbit column of SRAM local memory. IMAP uses an 8-bit processor formed around a carry look-ahead adder [189]. A shifter and look-up table are used to increase the performance of integer multiplications. With a bit-parallel PE, IMAP will achieve higher performance for applications with moderate to high parallelism compared to a bit-serial PE like the one in Terasys. VIRAM uses complex PEs and contains a 64-bit processor complete with floating-point operations [190]. The complexity of the PE limits the number of processors in the VIRAM-1 IC to four. Like the SIMD extensions added to conventional microprocessors, VIRAM allows each register in the 64-bit processor to be treated as packed data, and operations are permitted on eight 8-bit numbers, four 16-bit numbers, two 32-bit numbers, or one 64-bit number. C•RAM integrates many bit-serial processors at the memory sense amplifiers of a

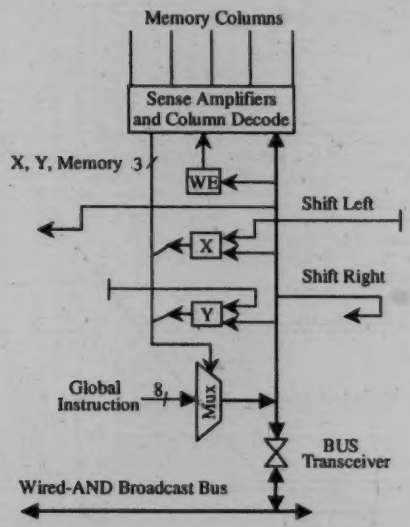

Figure 6.4: C•RAM processing element.

DRAM [191]. Embedded DRAM on the same IC allows a greater storage density than an SRAM-based design, at the cost of higher row access time latency. SRAM has the benefits of fast access time, does not require periodic refresh operations and can easily be implemented in a standard logic process. Figure 6.4 shows a simplified view of the C•RAM PE. The ALU is an 8×1 multiplexer, which allows 256 unique binary instructions to be applied to the eight data inputs. C•RAM has a simple linear network for inter-PE communication. Each processor can shift the result of an instruction to the right and store it in that neighbour's Y register, or shift it left and store it in that neighbour's X register. Once the data reaches the end of the array or the chip, rather than continuing in the left or right direction, the data can be shifted to a second C•RAM array. In the multiple linear array case, 2-D communication can thus be performed by a sequence of left or right shifts. The broadcast bus can be used to broadcast a single value to all PEs or can be used to perform a wired-AND operation with all PEs writing the bus. The wired-AND operation is useful for finding a global minimum value among the PEs. C•RAM uses a write-enable (WE) flag to conditionally enable individual PEs. Since a SIMD architecture requires each PE to execute identical instructions in lock-step, control structures that execute conditionally require a mechanism for disabling individual PEs. Nested control structures, such as the if-then-else clause, can be implemented using a stack of write-enable bits. C•RAM maps well into DRAM due to the simplicity of the PE and the narrowness of its layout. Since each PE

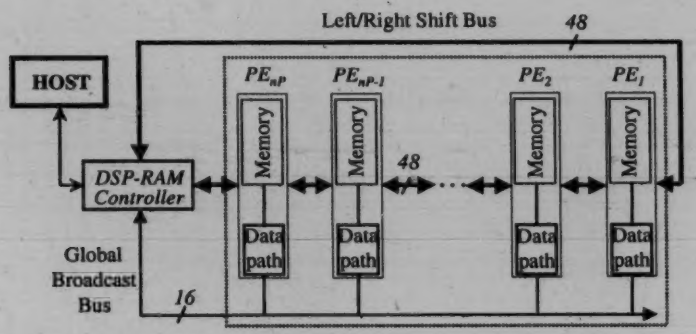

Figure 6.5: DSP-RAM architecture.

is very narrow, it is easier to pitch-match to the narrow DRAM memory columns than with a larger processing element. As well, many metal layers are not required for routing since there is less logic to contend with. Even so, DRAM-based C•RAM implementations still require each PE to be pitch-matched to some small number (e.g., 4) of memory columns rather than only one.

DSP-RAM is a processor-in-memory SIMD coprocessor architecture consisting of a linear array of n_P simple fixed-point PEs, as shown in Figure 6.5. The architecture was first developed at the University of Alberta and has been applied to a variety of different applications [165, 166]. Each PE consists of a data path, containing data registers and functional units, and a local memory. The DSP-RAM controller is a state machine that broadcasts micro-instructions to the PEs, exchanges data with the PEs (described later), and interfaces to the host processor. The DSP-RAM controller broadcasts one instruction stream and all PEs execute the same instructions in lock-step over multiple instances of data stored in their local memories. The DSP-RAM architecture is readily scalable from its HDL specification, and more processing elements can always be implemented as required to achieve any higher symbol decoding rate. The DSP-RAM architecture provides an efficient processing platform for implementing many algorithms with data-level parallelism [165, 166]. If the algorithm scales well on the linear array of the parallel architecture, the processing throughput will be increased directly by increasing the degree of parallelism (i.e., the number of PEs). More PEs implies more transistors on the chip. However, the processor can often operate at a slower clock frequency and still meet the required processing throughput. As well as lowering the clock frequency, one might also choose to lower the power supply voltage. Since power consumption is proportional to the square of the power

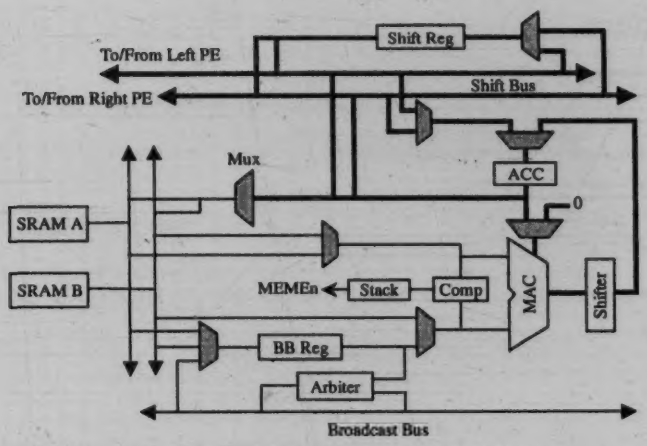

Figure 6.6: Architecture of one processing element.

supply [168], the possibility of reducing the voltage in DSP-RAM would be attractive for power-constrained processing platforms.

PIM-style architectures like DSP-RAM can offer the following advantages [167]:

- Internal memory accesses are usually much faster than external memory accesses.
- For high data rate applications, the restricted bandwidth to external memory tends increasingly to limit the overall performance. In the PIM-style architecture, the processor can directly-exploit the very large (e.g. 1024-bit) bus width at the sense-amplifiers of the internal memory blocks.
- Both the capacity and word width of custom on-chip memories is adjustable to any convenient value. There is no need to conform with standard memory configurations.
- System power consumption is reduced because fewer external memory accesses are generated. Such accesses consume significant power when driving the relatively high-capacitance of off-chip buses.

The architecture of one PE in DSP-RAM is shown in Figure 6.6. Each PE stores data in its own local memory, which we assumed to be partitioned into two banks, labeled SRAM A and B. Therefore two operands can be fetched simultaneously from memory. We assumed a word width of 16 bits but this can be easily adjusted. The core of each PE is a pipelined MAC. Two 16-bit operands can be multiplied and the 32-bit product can be added to the

content of the 48-bit accumulator (ACC) register through the adder/subtractor. Use of extended precision (48-bit) inner product accumulation reduces the rounding error and allows for more accumulation steps without the risk of overflow. Since the MAC is in the critical path, the number of pipeline stages in the MAC can be adjusted to achieve different throughputs. The shifter can perform a logical shift, an arithmetic shift or no shift before the MAC output is loaded into the ACC register. As an example, to execute the `MAC A, B` instruction two 16-bit operands are read from the memory banks and then multiplied in the MAC unit. Since the adder is enabled and the shifter is disabled by the controller, the 32-bit product is summed into the 48-bit accumulator value and stored. The result can then be kept in the accumulator, written back into the local memories, or written onto the shift bus to move it into the neighbor's PE shift register. To perform the division operation, rapidly-converging iterative algorithms based on multiplication can be used [105]. To perform division, the Goldschmidt algorithm takes advantage of the pipelined multiplier to permit division in $\lceil \log_2 16 \rceil = 4$ iterations [72].

It is possible to temporarily exclude processors in the DSP-RAM from executing an instruction depending on certain logical conditions. A comparator and stack are provided in each PE to support if-then-else conditional control flows. The depth of the hardware stack is configurable to allow different nesting depths in the program. The MEMEn signal enables the local memories in each PE. The arbiter module provides a global minimum compare operation (assuming a wired-AND implementation of the global broadcast bus) among all the PEs to speed up the merging of local results from the PEs into a single global result.

A simple communication network includes local left and right shift busses between adjacent PEs and a global broadcast bus. Data can be transferred into a PE in three ways: First, data can be transferred over the 16-bit wired-AND global broadcast bus from one data source (the host processor or one or more PEs) to all PEs. When the same constant needs to be stored into each PE, the broadcast bus can be used to broadcast the value into each PE's *broadcast bus register* (BBReg). For example, loading the $n_R \times n_T$ complex channel gain coefficients into the PEs local memories requires only $2n_R n_T + p$ clock cycles, where p is the number of pipeline stages between the broadcast bus and the local memory banks of the PE. Second, data can be sent to the left or right nearest-neighboring PEs over the 48-bit left/right shift bus, shown in bold arrows in Figure 6.6. The shift bus can be used to shift three 16-bit operands between PEs. Thus incoming ST symbols can be loaded efficiently

into the corresponding PE memories. Note that the operations of writing data into and reading data from the DSP-RAM can be interleaved. For example, previously decoded data may be shifted out to the host processor at the same time that a current block of data is being decoded. Since the I/O path over the shift bus uses only the 48-bit shift register and the shift bus, the memory is free during the shifting of incoming data and, therefore, significant power savings can be realized over a sequential memory load that requires a memory access for each data value [165]. Consequently, shifting of data can occur in parallel with the processing of local data in each PE. Finally, data can be loaded by memory write operations from the DSP-RAM controller or host processor into any locations in the two local memories. This requires one write instruction for every single data word and therefore is relatively slow.

6.5 Mapping LST Decoding onto DSP-RAM

One of the key decisions when implementing LST decoding on DSP-RAM is the mapping of subtasks onto the moderate number of PEs. Decomposing the decoding procedure into finer subtasks and distributing them among the PEs for parallel execution will affect the degree of concurrency and, consequently, the overall throughput. Moreover, since inter-PE communication delay often significantly affects the decoding throughput [44], if we map the calculation onto the PEs in such a way that inter-PE communication is minimized, we can generally expect higher performance and lower dynamic power consumption. Therefore the aim of the mapping is twofold: First, balancing the load among the PEs and increasing the resource utilization by uniformly distributing the decoding of the received ST symbols onto the available PEs. Second, according to the interconnection topology of the DSP-RAM, the decoding algorithm should be mapped onto the PEs to minimize the communication overhead.

We define a *thread* to be a recovery process for a ST symbol that was transmitted in the same symbol time by n_T transmitter antennas. As illustrated in Figure 6.7, a thread involves n_T nulling steps, n_T decoding steps and $n_T - 1$ cancellation steps. The recovery process can be completed entirely within a single symbol period or can be pipelined over a fixed number of symbol periods. For instance, as shown in Figure 6.7, a thread can be pipelined across several PEs (similar to the mapping used in [192]). Since all PEs execute the same stream of SIMD instructions, the last redundant cancellation step, shown with dotted lines in Figure 6.7, must also be performed. In addition, due to the data dependency

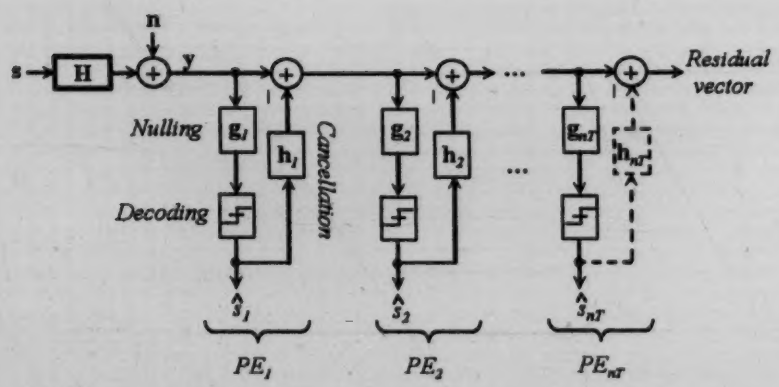

Figure 6.7: Dataflow diagram of the symbol decoding process for one ST symbol.

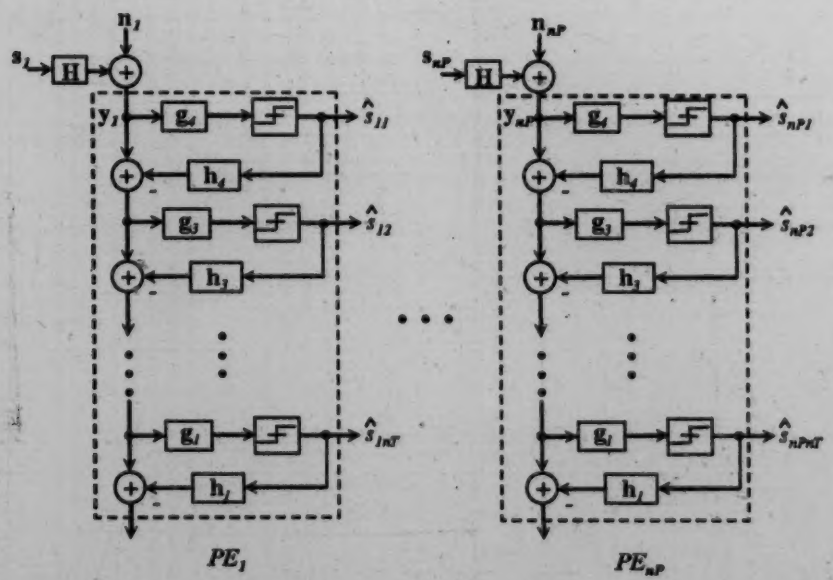

Figure 6.8: Dataflow diagram of the symbol decoding process for n_P ST symbols.

in the iterative three-step decoding algorithm, the communication penalty can be high. This particular mapping involves at most n_T PEs, implying that the maximum speed-up would be only n_T. Therefore, in this mapping the load cannot be uniformly distributed among only a moderate number (e.g. 64) of PEs. A more efficient approach, shown in Figure 6.8, takes advantage of the distributed memory architecture of the processor and maps a decoding thread to a dedicated PE. Thus the recovery of a ST symbol $\mathbf{y}^k$, where k is the k-th vector in a codeword matrix, can be performed by PE_k in the DSP-RAM. In this mapping the last additional cancellation step is not required. The mapping minimizes the inter-

PE communication and decouples the number of PEs in DSP-RAM from the number of transmitter/receiver antennas. Therefore, the number of PEs can be scaled based on the desired decoding rate. The greater the number of PEs, the greater the degree of parallelism and the higher the decoding throughput.

The decoding process starts by loading the channel matrix into each PE's local memory using the broadcast bus. For a Rayleigh block-fading channel model, the channel characteristics are assumed to be fixed over constant-sized L data blocks. Therefore the channel gains need to be loaded only once at the beginning of each block. In the next step, the nulling vector calculation can differ depending on the decoding algorithm. The MMSE nulling vectors can be efficiently obtained on DSP-RAM. First, the pseudo-inverse of the deflated channel matrix $\underline{\mathbf{H}}$, namely $\underline{\mathbf{H}}^{\dagger} = (\underline{\mathbf{H}}^H \underline{\mathbf{H}})^{-1} \underline{\mathbf{H}}^H$, has to be calculated n_T times at the beginning of each block. Since the dimensions of $\mathbf{H}$ are typically small, to obtain higher accuracy, we have chosen the direct approach [86] instead of the iterative method to invert the matrix.

There are several published techniques for calculating the matrix pseudo-inverse using direct routines [44]. In addition to the strong influence of the DSP-RAM microarchitecture on the complexity of the parallelized pseudo-inverse algorithm, the structure of the channel matrix $\mathbf{H}$ may also influence the choice of algorithm. Since the channel matrix in practice does not fall into any of the special cases, mapping Cramer's inverse method over $n_R \times n_T$ PEs gives an efficient parallel realization on DSP-RAM. This technique, which is also used in Intel's MMX library [193], requires only one division operation. To calculate the pseudo-inverse of n_T deflated channel matrices, $\underline{\mathbf{H}}$, with the rank ranging from n_T down to 1, n_T iterations are required. In iteration k, the k^2 cofactors of $\underline{\mathbf{H}}^H \underline{\mathbf{H}}$ can be calculated concurrently in complex arithmetic using a single instruction stream over k^2 PEs. Then cofactors are passed among the PEs and each PE calculates the $\underline{\mathbf{H}}^{\dagger} = \left[\frac{1}{det(\underline{\mathbf{H}}^H \underline{\mathbf{H}})} adj\,(\underline{\mathbf{H}}^H \underline{\mathbf{H}})\right] \underline{\mathbf{H}}^H$, where *det* and *adj* denote the determinant and the adjoint matrix operators, respectively. Note that assuming a block-fading channel model, the nulling vectors need to be calculated only once using n_T instances of pseudo-inversion of the deflated channel matrix $\underline{\mathbf{H}}$ at the beginning of each received block.

The process of loading the received ST symbols into the DSP-RAM can be overlapped with the nulling vector calculation. A block of received information is first divided into sub-blocks of size $n_T \times n_P$, where $n_P \ll L$, and the ST symbols of each sub-block are loaded consecutively using the left/right shift bus into the corresponding PE's memory. Since the

Table 6.2: Processor specifications.

Processor	Data Width	Instr. Width	Clock Freq. (MHz)	Core Volt. (V)	Core Power (mW)	Proc. Tech. (μm)
ARM7TDMI	16	16/32	133	1.08	332	0.13
SA-110	32	16	233	2.0	1000	0.35
PXA255	16/32	16/32	400	1.65	2598	0.18
TMSC6416T	8/16	32	720	1.2	2147	0.13
TMSC6713	32	32	225	1.26	1386	0.13
ADSP-TS203	32	32	500	1.05	2700	0.13

size n_P of a sub-block is typically much smaller than the block size L for the block-fading wireless channel model, this approach reduces the initial latency of the decoding process considerably.

After loading n_P ST symbols into the PEs' local memories and calculating the nulling vectors, each PE concurrently initializes the iterative three-step decoding process and decodes one received ST symbol. Therefore the time to process n_P threads is equivalent to the processing time of one single thread, hence the throughput is directly increased by a factor of n_P. When the parallel decoding is finished, the resulting symbols are shifted out from each PE into the host processor while the next set of n_P signals is shifted into the PEs' local memories. Therefore, the write and read operations are also overlapped. This process of decoding the received ST signals is continued until all of the received symbols in a block are recovered. Decoding of the next block can begin as soon as the channel gain coefficients are estimated at the receiver.

6.6 Implementation of a MIMO Receiver

To determine the efficiency and compare the performance of the parallel implementation of LST decoding on the moderately-parallel DSP-RAM architecture with conventional implementations on contemporary DSPs, a (4,4) MIMO system with 16-QAM modulation was modelled for six different processors. The processor specifications are summarized in Table 6.2. The ARM7TDMI is an embedded RISC processor with very low power consumption on a small die size that does not have any DSP-specific features [194]. The SA-110 uses architectural enhancements beyond the original ARM processor to execute at rates far exceeding those of the ARM7TDMI. The PXA255 processor from Intel Corp. is a 32-bit super-pipelined 16-bit SIMD processor intended to enhance audio/video decoder perfor-

Table 6.3: DSP-RAM PE implementation characteristics.

Device	*Clock Freq. (MHz)*	*Core Volt. (V)*	*Proc. Tech. (μm)*	*Slices*	*BRAMs*
Virtex E	98	1.8	0.18	713	2
VirtexII	120	1.5	0.13	684	2
VirtexII Pro	130	1.5	0.13	666	2

mance [195]. The VLIW-based TMS320C6416T-720 from Texas Instruments Inc. is a high-performance signal processor that contains two identical fixed-point data-paths [196]. The TMS320C6713-225 is a family of 32-bit floating-point DSPs that target applications such as 3-D graphics, radar and speech recognition. Analog Devices' ADSP-TS203 is optimized for demanding multiprocessor DSP applications such as communication infrastructure [50]. This processor supports both fixed-point and floating-point computations.

The regular architecture of FPGAs is a convenient platform for prototyping the linear DSP-RAM architecture. Contemporary FPGAs integrate megabytes of memory with multiple millions of equivalent logic gates arranged in a two-dimensional array of configurable logic slices [49]. A 64-PE DSP-RAM system was synthesized from a Verilog HDL description to various FPGAs. Each PE contained 512 bytes of local memory for implementations on the Virtex-E XCV3200E-7-FG1156, Virtex-II XC2V8000-5-FF1517 and Virtex-II Pro XC2VP125-5-FF1704 FPGAs. Table 6.3 shows the clock frequency and the resource utilization of the implementations of the DSP-RAM PE on the different FPGAs. For the FPGA implementations, dual-port BRAMs were synthesized and the maximum pipelined multipliers were implemented in LUTs. Since there are typically alternative arithmetic circuit implementations with different maximum clock rates and areas, one may choose different cells from the FPGA vendor's component library to meet the target application requirements. The same DSP-RAM design was also synthesized for a 0.18-μm TSMC CMOS technology standard cell implementation [197]. Figure 6.9 shows the layout of the 20.65 mm^2 64-PE DSP-RAM chip. The estimated core power consumption is 621-mW when the DSP-RAM operates at 100-MHz.

To develop the LST MIMO receiver algorithm, we used the following three-step design flow:

(1) MATLAB programs that model different LST decoding algorithms for an (n_T, n_R) MIMO system were written and then verified in simulation. The variance of the AWGN

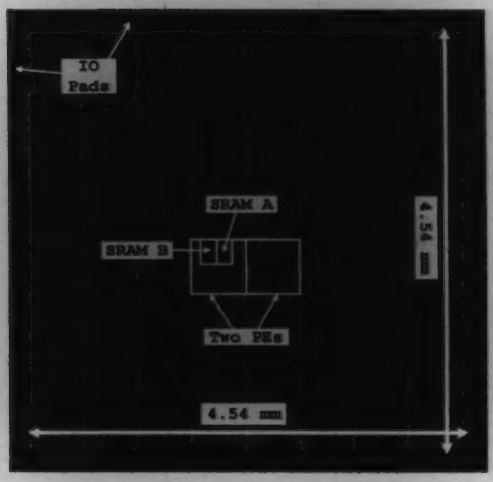

Figure 6.9: Chip Plot of a 64-PE DSP-RAM in 0.18-μm CMOS.

noise vector was normalized by

$$\sigma_n^2 = \frac{n_T \times E_s}{2 \log_2 q} \times 10^{-\rho_{\mathrm{dB}}/10}, \tag{6.14}$$

which is the noise energy per bit. The average energy per bit was thus normalized to 1. Here $\log_2 q$ is the number of transmitted bits per constellation point, ρ_{dB} is the SNR in dB, and $E_s = 2(q-1)/3$ is the mean symbol energy of the q-QAM constellation. Figure 6.10 plots the SER versus SNR simulation results for five different LST decoding algorithms for a (4,4) MIMO system that utilizes 16-QAM modulation over a flat Rayleigh fading channel. The ordered QR method requires one order of magnitude less computational complexity than the MMSE-LST decoding algorithm, but it has degraded SER performance due to the sub-optimal ordering employed in the modified Gram Schmidt calculation.

(2) Floating-point and fixed-point versions of the LST decoding algorithm were developed in C++, and these implementations were optimized and verified for six target processors. Considering the time and accuracy objectives, the LST decoding algorithm was expressed using complex arithmetic. Differences in the computer architecture, available programming languages, and compiler quality can make large differences in the way one implements a decoding algorithm. A parallel implementation of the LST decoding algorithm was developed on a DSP-RAM C++ emulator. Our emulator provides a debugging environment for a fixed-point implementation of the algorithm on the parallel DSP-RAM architecture and reports the exact clock cycle count required to execute a program, including the I/O and inter-PE communication cycles. This count can be used to choose

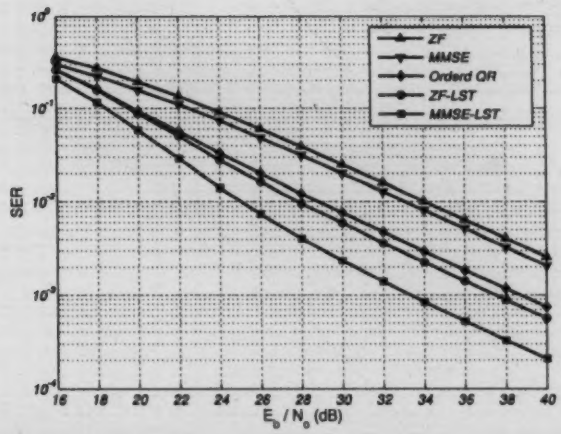

Figure 6.10: Symbol error rate vs. SNR for ZF, MMSE and three different LST algorithms, for a (4,4) MIMO system utilizing 16-QAM modulation over a Rayleigh flat-fading channel.

an appropriate clock frequency for the DSP-RAM to achieve real-time decoding. During implementation of the parallel LST decoding algorithm for the emulator, the developer is entirely responsible for controlling and efficiently utilizing the available functional units and for avoiding any possible structural hazards. The compiled code produced from a high-level language program implementation of LST decoding can be less efficient than expertly optimized assembly language code. Similarly, our optimized micro-coded implementation of LST decoding on DSP-RAM probably has an efficiency advantage over compiled code even when, as we did, all optimization features of the compiler enabled. Also, the PIM-style architecture of DSP-RAM requires many fewer clock cycles to access the on-chip memory banks than those required by conventional processors for off-chip memory accesses.

Table 6.4 gives the number of clock cycles required to decode received ST signals and the number of clock cycles required to calculate the nulling vectors for a 4×4 channel matrix. The reason that the number of clock cycles to decode 64 ST signals does not increase linearly with the number of ST signals is that most of the DSPs utilize some degree of parallelism in their instruction set architecture. For example, the ADSP-TS203 DSP provides a parallel core that can execute eight 16-bit MACs with 40-bit accumulation in one clock cycle. Also, the TMS320C6416T DSP contains two identical fixed-point data-paths. DSP-RAM can provide a much greater degree of parallelism (e.g., $n_P = 64$) than these DSPs and, therefore, the data-parallel LST decoding algorithm can scale efficiently on the linear architecture of DSP-RAM. Thus the number of clock cycles to decode 64

Table 6.4: Cycle counts to decode received ST signals and calculate nulling vectors of a 4×4 channel matrix.

Processor	*Decode One ST Symbol*	*Decode 64 ST Symbols*	*Calc. Nulling Vectors*
ARM7TDMI	4,798	206,396	43,641
SA-110	3,124	146,698	33,701
PXA255	3,732	155,602	22,698
TMSC6416T	1,812	87,714	25,340
TMSC6713	1,814	90,386	17,572
ADSP-TS203	1,555	70,455	13,216
DSP-RAM	1,078	1,174	12,742

ST symbols is not much more than the number required to decode one received signal and hence the throughput is increased by a factor of 64. To calculate the 4 nulling vectors of a 4×4 channel matrix, one must perform four pseudo-inversions of the reduced channel matrices, with the rank reducing successively from 4 to 1. For a DSP-RAM implementation, the pseudo-inverse operation can be performed efficiently in parallel using the mapping described in Section 6.5 over 16 PEs. However, since the required degree of parallelism is only $n_R \times n_T$, the number of clock cycles required to calculate 4 nulling vectors in a DSP-RAM implementation is close to the number required by the slightly parallel ADSP-TS203 DSP.

(3) Once the design was completed on the emulator, test vectors were designed and used as the stimulus to the HDL model. After the algorithm was verified in simulation, it was synthesized for the target FPGA. Figure 6.11 plots the SER versus SNR results of the ordered QR-LST decoding for the various number representations of the MIMO system reported in Figure 6.10. The product word length and sum word length were set to 32 and 48, respectively, for all implementations. Note that a 12-bit fraction field, labeled as $FI(16, 12)$ in the figure, achieves almost the same SER performance as the floating-point implementations up to a SNR of 32 dB.

Consider a (4,4) MIMO system that exploits spatial multiplexing using an LST architecture and 16-QAM modulation. We will make the common assumption of symbol-synchronous receiver sampling and ideal timing recovery. Also, for a typical indoor wireless environment with a maximum Doppler frequency of $f_D = 3\ Hz$, the coherence time of the channel can be calculated as $T_c = 0.423/f_D$ [115]. Assuming a block-fading wireless channel, where the channel response is almost invariant during the coherence time of the

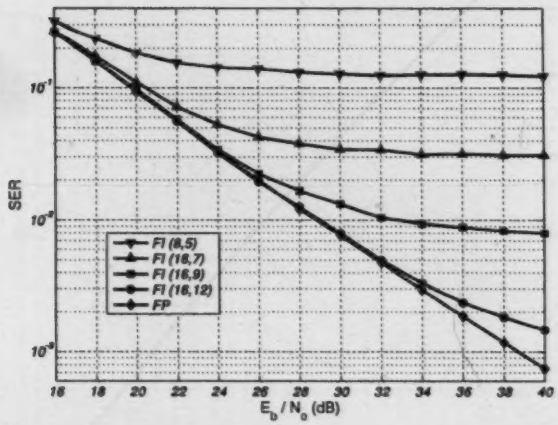

Figure 6.11: SER vs. SNR for floating-point (FP) and four fixed-point number representations.

channel, the block duration can be chosen to be $T_b = 100$ ms. Let k_n denote the number of clock cycles required to compute n_T nulling vectors. To achieve real-time decoding, $\frac{k_n+(L_d \times k_d)}{f_p}$ should be less than or equal to T_b, where k_d is the number of clock cycles required to decode one received ST signal, L_d is the number of ST data symbols in a block of length L, and f_p is the clock frequency of the receiver signal processor. Assume that each block is divided into a set of 64 ST symbols frames. Since most conventional processors utilize some degree of parallelism in their instruction set architecture, we introduce k_{d64} to denote the number of clock cycles required to decode one frame of 64 ST symbols. Therefore, the number of ST symbols in a block is given by:

$$L_d = \frac{K - k_n}{k_{d64}} \times 64 \tag{6.15}$$

where $K = T_b \times f_p$ is the total clock cycle budget for T_b periods. Equation 6.15 shows that in addition to the clock frequency of the signal processor, the efficiency of the processor when executing the LST decoding algorithm has a great influence on decoding performance. To maximize the decoding throughput, the denominator of Equation 6.15 should be minimized. An efficient way to achieve this goal is to map the decoding algorithm over an array of PEs that operate concurrently. The results in Table 6.5 show that a 64-PE DSP-RAM can decode at more than 10 times the bit rate of a high-performance conventional DSP processor. The higher throughput is achieved even though DSP-RAM's assumed 100-MHz clock frequency is much slower than that of the 720-MHz TMSC6416T DSP.

Table 6.5: Decoding throughput of the LST decoder implementations for a (4,4) MIMO system utilizing 16-QAM modulation, with the block-fading channel model assumption, and $T_b = 100$ ms.

Processor	Clock Freq. (*MHz*)	Max. Decoding Throughput (*Mb/s*)
ARM7TDMI	133	0.64
SA-110	233	1.58
PXA255	400	2.56
TMSC6416T	720	8.2
TMSC6713	225	2.48
ADSP-TS203	500	7.09
DSP-RAM	100	85.07

6.7 Conclusions

Multiple antenna communication systems can achieve remarkably high data rates with no increase in bandwidth or transmitted power; however, symbol decoding in a MIMO receiver is a computationally-intensive process. Optimal or exact decoding algorithms, such as ML, require time that is exponential in the number of symbols that must be considered. A less computationally-intensive method, such as the SD algorithm, attempts to prune the search space and thereby provide substantial computational saving over ML decoding. However, the SD algorithm has an irregular control sequence and hence is inefficient for mapping on a parallel processor architecture. When the channel is perfectly known to the receiver, the heuristic LST decoding scheme can be used efficiently to decode received signals with realizable computational complexity. Due to the inherent data parallelism in the LST decoding algorithm, as an efficient alternative to using a single, high-performance processor to achieve real-time decoding, multiple simpler processors can be used to exploit the available data-level parallelism.

A PIM-style moderately-parallel architecture, called DSP-RAM, has been synthesized to implement parallel LST MIMO receiver algorithms. DSP-RAM is a lightweight parallel computing architecture that combines an array of simple fixed-point datapaths with local SRAMs to provide high-performance signal processing in a power-efficient core. This configuration exposes and exploits the large internal bandwidth that is available collectively at the SRAMs, allowing DSP-RAM to outperform conventional high-performance microprocessors and DSPs for a variety of important moderately-parallel algorithms. The performance results demonstrate that a 64-PE 100-MHz DSP-RAM can potentially provide more

than 10 times greater decoding throughput in a typical indoor environment compared to a high-performance 720-MHz DSP processor. The significant speed-up of the parallel DSP-RAM architecture can be exploited to permit a lower operating clock frequency and/or a lower operating voltage, which would have the further benefit of lowering the power consumption. Since the structure of LST decoding algorithms scales very well on the linear array of PEs in DSP-RAM, the data throughput can be readily increased by using more PEs.

Chapter 7

Conclusions and Future Work

Wireless communication systems share one common challenge: they all must operate over a multipath fading channel. A vast number of mathematical representations of the time-varying wireless channel have been proposed. Surprisingly, some of the best-known models have been used extensively over decades for the emulation and performance evaluation of communication systems do not in fact faithfully reproduce characteristics of real-world channel conditions. It is therefore essential to carefully evaluate the statistical properties of any channel model that is being considered as the basis of a fading channel simulator.

In addition to channel impairments, noise at the receiver can impact the performance of communication systems. Such noise is commonly modeled as variates with a Gaussian distribution due to the Central Limit Theorem. When evaluating the physical layer algorithms that operate at a very low error rate, the value of the noise samples at the tails of the Gaussian distribution will be the dominant source of errors. Since the tail of Gaussian distribution decays near exponentially, generating Gaussian variates (GVs) with large values is quite challenging.

Monte Carlo (MC) simulation of wireless communications relies on the accuracy of the *additive white Gaussian noise* (AWGN) generator and the multipath fading channel model. Purely software MC simulation of physical layer algorithms of wireless communication systems that operate at very low error rates is becoming prohibitively long. Fortunately, digital baseband simulators provide several orders of magnitude speedup over software based simulators, thereby greatly accelerating the iterative process of product design and evaluation, and ultimately reducing the time-to-market for communication devices.

We addressed the above challenges first by designing and implementing an accurate and compact digital Gaussian variate generator on field-programmable gate arrays (FP-

GAs). The AWGN core can be easily configured to achieve high tail accuracy, much higher than is available in most commercial hardware-based noise generators. The rate that GVs can be generated is limited by the rate of a pipelined multiplier on FPGAs. Second, several hardware-based digital baseband multipath fading channel simulators were designed and evaluated. An improved fading simulator based on the sum-of-sinusoids (SOS) technique is proposed that shows better statistical properties to Clarke's reference model compared to the published models. Also, a novel design technique was used to alleviate the complexity of fading channel simulators by co-designing the required digital filters. While other realizations of channel emulators use a heterogeneous architecture (usually consisting of DSPs, FPGAs, etc.) to implement the required computationally-intensive multi-rate signal processing algorithms of filter-based techniques, our implementation uses only a small fraction of a single FPGA. The ability to implement an entire digital baseband fading channel emulator along with AWGN generator on a small fraction of a single FPGA should be a significant improvement for the rapid prototyping and verification of wireless systems.

Integration of programmable fabrics with a moderate number of memory blocks in an FPGA provides an efficient platform for realizing parallel *processor-in-memory* (PIM) style processor architectures. A scalable moderately-parallel signal processor architecture that combines the *single-instruction multiple-data* (SIMD) and PIM approaches, called DSP-RAM, was designed and implemented on FPGAs. Our implementation results verified that DSP-RAM can efficiently increase the performance of layered space-time decoding algorithms for the spatial multiplexing scheme of *multiple-input multiple-output* (MIMO) communication systems compared to currently-available high-performance, but power-hungry and costly, DSPs. DSP-RAM can be used as a co-processor to a conventional microprocessor to increase the throughput of computationally-intensive algorithms that can be mapped efficiently on the data-parallel architectures.

In summary, this thesis made contributions in four areas:

- Design and implementation of the most compact and fastest disclosed digital Gaussian variate generator (GVG) with accurate statistical properties. The GVG occupies only 1% of a single Xilinx Virtex-II XC2V4000-6 FPGA and operates at 253 MHz [9], generating 506 million GVs per second within a range of $\pm 9.41\sigma$. The design can be configured to achieve higher tail accuracy at a small cost in extra hardware but with slightly decreased operating rate.

- In addition to two compact implementations of a SOS-based fading channel simulator,

an improved fading channel model based on the SOS approach is described. The proposed model improves the statistical properties of generated fading variates compared to previously proposed models. A fixed-point implementation of the fading channel simulator on a FPGA utilizes only 5% of the configurable resources and generates over 200 million 16-bit fading variates per second.

- A much more compact and yet accurate implementation of a parameterized fading channel simulator using digital infinite-duration impulse response (IIR) filters is described. A novel filter design scheme is proposed to implement the shaping filter and the interpolation low-pass filters together on a single FPGA. The new design is the first digital baseband fading channel simulator that is realizable on a fraction of a single FPGA. The fixed-point implementation of Rayleigh fading channel simulator on an FPGA utilizes only 4% of the configurable slices, 20% of the dedicated multipliers and, 2% of the available memories on a Xilinx Virtex2P XC2VP100-6 FPGA, while generating 25 million fading variates per second. The parameterized mobile channel simulator can be reconfigured to accurately simulate a wide variety of different channel characteristics. Also, a filter processor with a very short instruction set is proposed to be controlled by a micro-program. A micro-programmed controller makes debugging and scaling of the fading channel simulator much easier compared to modifying the control unit with random logic to support MIMO and frequency-selective channels.

- An existing moderately-parallel and scalable architecture, called DSP-RAM, that combines the single-instruction multiple-data (SIMD) and *processor-in-memory* (PIM) approaches to increase the performance of moderately data-parallel signal processing applications is applied efficiently to the MIMO signal decoding problem. Integrating simple fixed-point datapaths, also called processing elements (PEs), with the memories exposes the enormous data bandwidth between the two, and eliminates the bottleneck that otherwise occurs on an external bus between the memory chips and processor(s) in conventional architectures. The DSP-RAM architecture can be readily synthesized and mapped to standard FPGAs. By efficiently mapping the layered space-time (LST) MIMO algorithm onto the DSP-RAM architecture, it is shown that for a typical indoor wireless environment, a 100-MHz DSP-RAM can potentially provide more than 10 times greater decoding throughput at the receiver of a (4,4) MIMO system compared to a conventional 720-MHz DSP. The degree of parallelism (i.e., the number of PEs) can be easily scaled up to increase the throughput of a parallel algorithm. Also, one has the option of using increased parallelism

to run at a slower clock frequency to simplify the implementation and still meet the required processing performance.

7.1 Future Work

Future work could be pursued in a number of directions as described in the following sections. An alternative GVG using faster iterative algorithms is proposed in Section 7.1.1. The proof of the ergodicity of the proposed model in Chapter 4 is also considered as a future work and is discussed in Section 7.1.2. Section 7.1.3 presents channel simulators for frequency-selective fading channels and MIMO channels. Section 7.1.4 presents a joint channel estimation and symbol decoding scheme for an LST architecture.

7.1.1 An Alternative Gaussian Variate Generator

7.1.1.1 A GVG with Less Residual Error

The proposed GVG in Chapter 3 uses piecewise polynomial curve fitting. While the magnitude of generated GVs are important when modeling additive noise at the receiver, the machine precision-level fitness of the PDF of generated GV with ideal Gaussian PDF is not crucial. One may be more interested in the numerical accuracy of the generated GVs and in minimizing the residual error between the Gaussian PDF and the PDF of generated noise samples. In this case, two key points that could be considered are (1) increasing the number of segments for non-uniform segmentation of $f(u_1)$ and/or using a higher order degree polynomial for curve fitting approximation; and (2) similarly, a more accurate approximation of $g(\cdot)$ could be obtained using a higher-order polynomial and non-uniform quantization of trigonometric functions.

Some of our initial work on this approach is shown in Figure 7.1. In this case, $f(u_1)$ is segmented uniformly on a logarithmic scale into 11 segments denoted by s_i, $i = 0, \ldots, 10$, where segment s_i represents the interval $[2^{-3i}, 2^{-3(i+1)}]$. Note that with u_1 represented in fixed-point format $\mathcal{Q}\langle 32, 31\rangle$, segment s_{10} corresponds to the interval $[2^{-31}, 2^{-30}]$. When u_1 approaches 0, where $u_1 \in [2^{-3}, 2^{-31}]$, the very small value of u_1 resides in one of the segments $s_1, \ldots, s_{10}$. When u_1 approaches 1, where $u_1 \in (2^{-3}, 1)$, the gradient of $f(u_1)$, $\partial f(u_1)/\partial u_1$, tends toward infinity. For greater accuracy, segment $s_0 \in (2^{-3}, 1)$ is thus subdivided nonuniformly into six segments, $s_{11}, \ldots, s_{16}$, as shown in Figure 7.2.

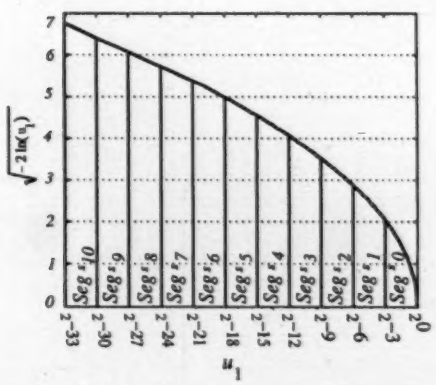

Figure 7.1: Logarithmic segmentation of the domain of $f(u_1)$.

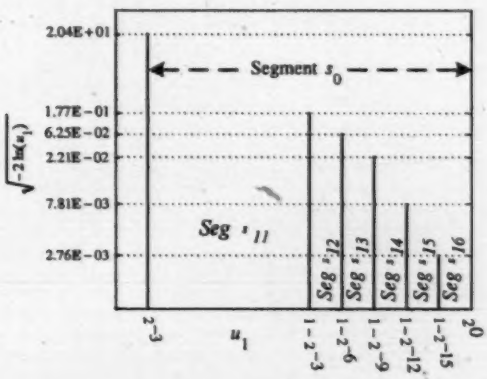

Figure 7.2: Sub-segmentation of s_0.

7.1.1.2 A Faster Iterative Gaussian Variate Generator

While CORDIC method and *additive-normalization* techniques explained in Chapter 3 can be used to compute $\ln(\cdot)$ and square-root operations, other schemes for evaluating these operations can be utilized to implement the Box-Muller algorithm. These alternative schemes may have advantageous over other proposed algorithms with certain implementation techniques or availability of specific features in target hardware platform.

One approach to calculate $\ln(x)$, where $x \in [1,2)$, is based on *multiplicative normalization* where multiplication is done by shift and add operations:

$$\begin{aligned} x_{i+1} &= x_i c_i = x_i(1 + 2^{-i}d_i), \quad d_i \in \{-1, 0, 1\} \\ y_{i+1} &= y_i - \ln c_i = y_i - \ln(1 + 2^{-i}d_i) \end{aligned}$$

where $\ln(1 + 2^{-i}d_i)$ is read out from a table [105]. If we start with $x_0 = x$, $y_0 = y$, and

choosing d_i digits such that x_m converges to 1, after m steps:

$$\begin{aligned} x_m &= x\prod c_i \approx 1 \Rightarrow \prod c_i \approx 1/x \\ y_m &= y - \sum \ln c_i = y - \ln \prod c_i \approx y + \ln x. \end{aligned}$$

Starting with $y = 0$ leads to the computation of $\ln x$. In order to calculate $\ln x$ for any x outside $[1, 2)$, x can be written as $x = 2^\eta s$ with $s \in [1, 2]$. Then $\ln x = \ln s + \eta \ln 2 = \ln s + 0.693147180\eta$.

In the above algorithm, to obtain $\ln x$ with k bits of precision, k iterations are required, which is still relatively slow. A faster algorithm is proposed by Lo [198] that requires a multiplier to compute $\log_2 x$ where $x \in [1, 2]$. Calculating the natural logarithms can be easily done by scaling base-2 logarithms. Let $y = \log_2 x$ be a fractional number in binary as $(.y_{-1}y_{-2}\cdots y_{-k})_2$. Hence, $x = 2^y$ and $x^2 = 2^{(y_{-1}.y_{-2}\cdots y_{-k})_2}$. If $y_{-1} = 1$ then $x^2 \geq 2$, thus computing x^2 and comparing the result with 2 determines the MSB y_{-1} of y. When $y_{-1} = 1$, then $x^2/2 = 2^{(1.y_{-2}y_{-3}\cdots y_{-k})_2}/2 = 2^{(.y_{-2}y_{-3}\cdots y_{-k})_2}$. Subsequent bits of y can be determined in a similar way. Algorithm 8 can be used to calculate $\log_2 x$ for $x \in [1, 2]$:

Algorithm 8 Calculating $\log_2 x$, $x \in [1, 2)$

```
for (i = 1; i == l; i++) do
  x = x^2;
  if (x ≥ 2) then
    y_{-i} = 1; x = x/2;
  else
    y_{-i} = 0;
  end if
end for
```

The iterative Newton-Raphson algorithm [105] can be used to approximate the square root function. This algorithm is based on a general method to obtain the zero of the function (i.e., the value of x for which $f(x) = 0$). The recurrence equation can be written $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, where $f'(x_n)$ denotes the derivative of the function $f(x)$ evaluated at x_n. To compute $\sqrt{d}$ using the Newton-Raphson method, $f(x)$ can be chosen as $f(x) = x^2$ which has a root at $x = \sqrt{d}$. The function $f(x) = x^2 - d$ leads to $x_{i+1} = 0.5(x_i + d/x_i)$. In the case of fractional square rooting, where $d \in [0.5, 1)$, $(1 + d)/2$ provides a good starting value [105]. An alternative approach that avoids the division operation in the previous recurrence that has found wider application is based on computing the reciprocal $\sqrt{d}$ then multiplying the result by d [72, 105]. We can use the function $f(x) = 1/x^2 - d$ that has a root at $x = 1/\sqrt{d}$ and get the recurrence $x_{i+1} = 0.5x_i(3 - dx_i^2)$. Each iteration

now requires three multiplications and one addition, versus one digit selection, one digit multiplication, and one addition in the digit recurrence method. However, the quadratic convergence requires only a few iterations with a suitably accurate initial estimate. For example if x_1 is accurate within half the machine precision, a second iteration to find x_2, followed by a multiplication by d, completes the process. For subtraction from 3, we can use a bit inversion since $3 - dx_i^2 = 1 + (2 - dx_i^2)$ where the term $2 - dx_i^2$ corresponds to a two's complement, which can be approximated by a bit inversion.

7.1.2 On the Ergodicity of the SOS-based Fading Channel Models

The proof of the ergodicity of the model proposed in Section 4.6 in Chapter 4, namely *Model II* can be considered as a future work. The ACF and CCF of the quadrature components of the proposed fading signal can be written as:

$$\begin{aligned}
\mathrm{R}_{c_i,c_i}(\tau) &= \mathrm{E}\{c_i(t)c_i(t+\tau)\} \\
&= \frac{2}{M}\sum_{n=1}^{M}\sum_{i=1}^{M}\mathrm{E}\Big\{[\cos(\omega_d t\cos(\alpha_n(t))\cos(\varphi_n(t)) - \\
&\quad \sin(\omega_d t\cos(\alpha_n(t))\sin(\varphi_n(t))]\times \\
&\quad [\cos(\omega_d(t+\tau)\cos(\alpha_i(t+\tau))\cos(\varphi_i(t+\tau)) - \\
&\quad \sin(\omega_d(t+\tau)\cos(\alpha_i(t+\tau))\sin(\varphi_i(t+\tau))]\Big\} \\
&= \frac{2}{M}\sum_{n=1}^{M}\mathrm{E}\Big\{\cos\left(\omega_d t\cos(\alpha_n(t)) + \right. \\
&\quad \left.\psi_n(t)\right)\cos\left(\omega_d(t+\tau)\cos(\alpha_n(t+\tau)) + \psi_n(t+\tau)\right)\Big\} \qquad (7.1)
\end{aligned}$$

$$\begin{aligned}
\mathrm{R}_{c_q,c_q}(\tau) &= \mathrm{E}\{c_q(t)c_q(t+\tau)\} \\
&= \frac{2}{M}\sum_{n=1}^{M}\sum_{i=1}^{M}\mathrm{E}\Big\{[\cos(\omega_d t\sin(\alpha_n(t))\cos(\psi_n(t)) - \\
&\quad \sin(\omega_d t\sin(\alpha_n(t))\sin(\psi_n(t))]\times \\
&\quad [\cos(\omega_d(t+\tau)\sin(\alpha_i(t+\tau))\cos(\psi_i(t+\tau)) - \\
&\quad \sin(\omega_d(t+\tau)\sin(\alpha_i(t+\tau))\sin(\psi_i(t+\tau))]\Big\} \\
&= \frac{2}{M}\sum_{n=1}^{M}\mathrm{E}\Big\{\cos\left(\omega_d t\sin(\alpha_n(t)) + \right. \\
&\quad \left.\psi_n(t)\right)\cos\left(\omega_d(t+\tau)\sin(\alpha_n(t+\tau)) + \psi_n(t+\tau)\right)\Big\} \qquad (7.2)
\end{aligned}$$

$$\mathrm{R}_{c_i,c_q}(\tau) = 0, \quad \mathrm{R}_{c_q,c_i}(\tau) = 0$$

When $\alpha_n(t+\tau) = \alpha_n(t)$, $\varphi_n(t+\tau) = \varphi_n(t)$, and $\psi_n(t+\tau) = \psi_n(t)$, then the modified model reduces to the model in (4.10). Let define the time average correlations as

$$\begin{aligned}
\tilde{\mathrm{R}}_{c_i,c_i}(\tau) &= \lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T} c_i(t)c_i(t+\tau)\mathrm{d}t \\
\tilde{\mathrm{R}}_{c_i,c_q}(\tau) &= \tilde{\mathrm{R}}_{c_q,c_i}(\tau) = 0 \\
\tilde{\mathrm{R}}_{Z,Z}(\tau) &= \frac{1}{2}\lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T} Z^*(t)Z(t+\tau)\mathrm{d}t
\end{aligned} \tag{7.3}$$

To proof the ergodicity of the proposed *Model II*, we need to show that

$$\begin{aligned}
\tilde{\mathrm{R}}_{c_i,c_i}(\tau) &= \mathrm{R}_{c_ic_i}(\tau) = \mathcal{J}_o^2(2\pi f_D\tau) \\
\tilde{\mathrm{R}}_{c_i,c_q}(\tau) &= 0 \\
\tilde{\mathrm{R}}_{Z,Z}(\tau) &= \mathrm{R}_{Z,Z}(\tau) = 2\mathcal{J}_o^2(2\pi f_D\tau)
\end{aligned} \tag{7.4}$$

The variance of the time average, $\mathrm{Var}\{\mathrm{R}_{c_i,c_i}(\tau)\} = \mathrm{E}\Big[|\tilde{\mathrm{R}}_{c_i,c_i}(\tau) - \mathcal{J}_o^2(2\pi f_D\tau)|^2\Big]$, can be considered as a statistical measure for closeness of statistics between a single trial with finite number of sinusoids N and the ideal case with $N = \infty$ [37].

7.1.3 Architecture of a Wireless Channel

As discussed in Chapter 4 and Chapter 5, the complex envelope of the fading signal is modeled as a complex Gaussian process at the receiver of a wireless channel. There are two common cases that must be considered when simulating a realistic wireless channel:

- A primary effect of wireless channel environment on a transmitted electromagnetic signal is that it loses its energy density due to interactions with the propagation environment. The difference between the transmitted signal power and the received power is called the *path loss* and denoted by PL. While the local mean of the small-scale fading process $c(\tau, t)$ is approximately constant for small distances, it can vary considerably over large distances. *Shadow fading* is another effect of wireless channel that can be characterized as a *multiplicative* process of multiple reflections in a multipath environment. By virtue of the CLT, multiplication of various path amplitudes due to multiple reflections per path results in a lognormal distribution. Thus shadow fading can be described as a random variable with Gaussian distribution (with values in dB) about the mean of path loss. While shadow fading varies faster than path loss, both effects vary more slowly than small-scale fading. This slow variation of the mean received signal strength over large distances or long time intervals is known as *large-scale* effects and denoted by $\gamma(t)$.

• When a strong LOS path exists in addition to the scattered paths, then the process has a non-zero mean (arising from the LOS signal) and the magnitude of the process becomes *Ricean* [23]. This strong component may be a LOS path or a path that goes through much less attenuation compared to the other received components (also called a *non-faded* or *specular* path). The Rician PDF is often characterized by the ratio of the power of the direct component to the power of the scattered component $K(\text{dB}) = 10\log_{10} K$ where the ratio K is called the *Ricean factor*. In the presence of a specular path, the fading signal $g(\tau, t)$ can be considered to be the sum of two components: a Rayleigh component $c(\tau, t)$ and a deterministic (in amplitude and phase) component $d(t)$ representing the LOS path as

$$g(\tau, t) = \sqrt{\frac{1}{K+1}}\, c(\tau, t) + \sqrt{\frac{K}{K+1}}\, d(t) \tag{7.5}$$

where K is the Rician factor and $d(t)$ can be written as $d(t) = ae^{j\omega_d t + \phi_d}$ where the amplitude, Doppler shift, and phase of the LOS component are denoted by a, ω_d, and ϕ_d, respectively [199]. If the Doppler shift along the LOS path is zero, then the mean value $d(t)$ is time-invariant.

Signal attenuation in a radio channel is commonly represented as the product of large-scale and short-scale fading as $h(\tau, t) = g(\tau, t) \times \gamma(t)$, where $h(\tau, t)$ is the mobile channel impulse response in its complex-lowpass form. The fading channel model can be scaled to extend the number of independent channels to simulate n_T-input, n_R-output MIMO channels and other diversity schemes. For example, Figure 7.3 shows a MIMO channel with n_T transmit antennas and n_R receiver antennas. The parameters of each channel can be configured separately to be able to simulate the multiple antenna channels. The Python filter processor described in Chapter 5 can be efficiently configured to simulate a narrow-band or wide-band MIMO channel.

While the above model assumes that different fading sequences are correlated in time but uncorrelated in space, in a real world scenario, the fades are usually not independent and exhibit spatial correlations between sequences. The fading correlation depends on the physical parameters of the multi-element antenna (e.g., antenna spacing) and on the scatter characteristics (e.g., lack of independent propagation paths). As discussed in Section 5.2 in Chapter 5, in order to obtain the space-time correlation characteristics for a given path, a temporal correlated random process can be followed by a linear transformation to become also spatially correlated.

Assume $\mathbf{\Psi}_T$ and $\mathbf{\Psi}_R$ are the long-term stable transmitter and receiver correlation ma-

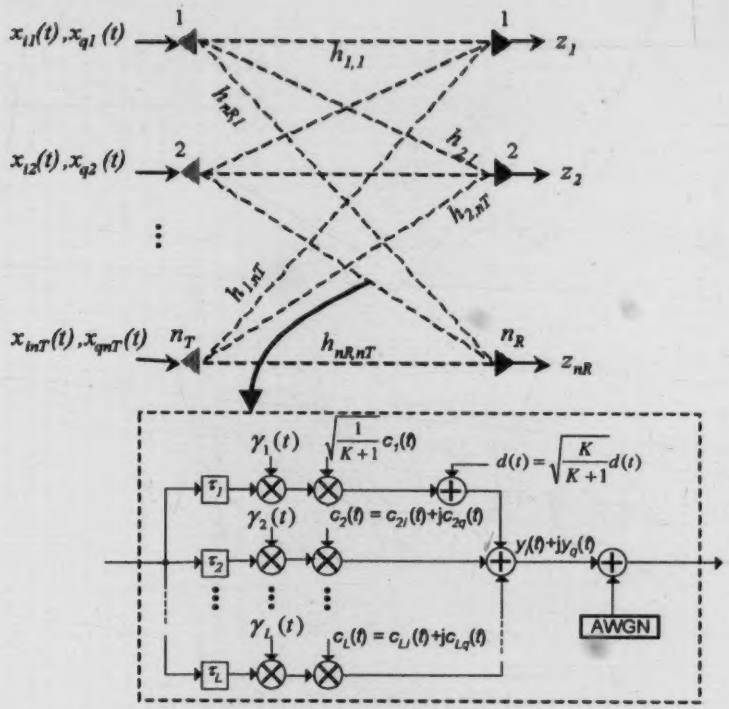

Figure 7.3: MIMO channel simulator.

trix, respectively. If $\mathbf{H}_w$ denotes a $n_R \times n_T$ matrix of i.i.d Gaussian variables of unity variance, then the correlated MIMO channel model can be written as $\mathbf{H} = \mathbf{A}^H \mathbf{H}_w \mathbf{B}$ [200], where $\mathbf{A}$ and $\mathbf{B}$ are obtained using Cholesky decomposition of $\Psi_T = \mathbf{B}\mathbf{B}^H$ and $\Psi_R = \mathbf{A}\mathbf{A}^H$, respectively. Usually the transmitter is assumed to be elevated and unobstructed while the receiver is taken to be surrounded by a scattering ring. Thus the receive correlations are much smaller than the transmit ones.

Figure 7.4 plots our initial simulation results for the SER versus SNR for a (4,4) MIMO system correlated only in time and correlated in time as well as space using a 4-QAM modulation and $f_D T_S = 0.02$. The element-wise absolute value of transmitter correlation matrix used in the simulation is:

$$\text{abs}(\Psi_T) = \begin{pmatrix} 1.000 & 0.626 & 0.620 & 0.601 \\ 0.626 & 1.000 & 0.626 & 0.620 \\ 0.620 & 0.626 & 1.000 & 0.626 \\ 0.601 & 0.620 & 0.626 & 1.000 \end{pmatrix}$$

As the plot in Figure 7.4 shows, fading correlation weakens the advantage of diversity and results in a performance loss.

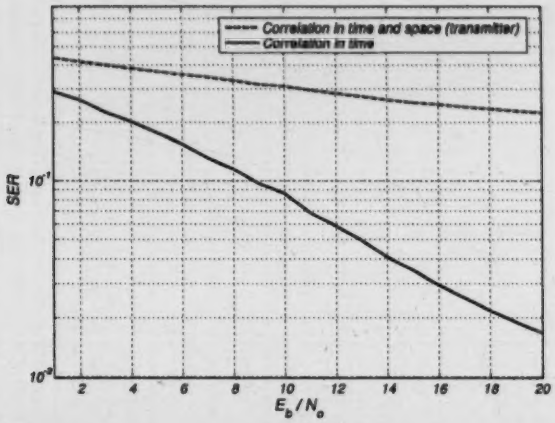

Figure 7.4: SER versus SNR for (4,4) MIMO system with a 4-QAM modulation. (b): The system is correlated in time. (a) The system is correlated in both time and space.

7.1.4 Joint Channel Estimation and Symbol Decoding for LST Decoders

High-speed data services such as BLAST generally target low mobility users, where the channels are slow-fading. An optimal channel estimator for continuous fading channels should account for the structure of the channel variation. When transmitting a block of L ST symbols, the relation between the input and output signals of a narrow-band MIMO link is represented in the equivalent discrete time baseband model by the complex vector notation

$$\mathbf{Y} = \sqrt{\frac{\sigma^2}{n_T}}\,\mathbf{H}\,\mathbf{S} + \mathbf{N} \tag{7.6}$$

where $\mathbf{S}$ is the codeword matrix where rows correspond to different transmitter antennas and columns correspond to different times. It is assumed that the entries of transmitted signal matrix $\mathbf{S}$ have unit mean-square. Thus, the average total transmit power σ^2 becomes the expected SNR at each receiver antenna. $\mathbf{H}$ is the $n_R \times n_T$ matrix of fading path gains, and $\mathbf{N}$ is an $n_R \times L$ matrix of additive noise samples. The matrices $\mathbf{H}$ and $\mathbf{N}$ both comprise independent zero mean unit variance complex Gaussian entries. The $n_R \times L$ received signals $\mathbf{Y}$ are corrupted by additive noise that is statistically independent among the n_R receivers and the L symbol periods. In the training-based CSI estimation, the matrix $\mathbf{S}$ consist of both known and unknown symbols. The known symbols are used for estimating the unknown channel at the receiver and the unknown symbols represent the transmitted information symbols. Therefore, the problem can be partitioned into a channel estimation step, followed by a data detection conditioned on the estimated channel. If the training

occupies L_τ symbols and data transmission occupies L_d symbols, the time $L = L_\tau + L_d$ is referred to as the *channel coherence time*. In a time-varying flat-fading channel, the actual channel will deviate progressively from the initial channel estimate obtained at the beginning of each received block L. Two successive channel realizations at the beginning of each block are independent of each other and, therefore, the channel is called memoryless.

Discrete channel estimation comprises of two phases. In the training phase, the base-band model of system can be written as

$$\mathbf{Y}_\tau = \sqrt{\frac{\sigma_\tau^2}{n_T}}\,\mathbf{H}\,\mathbf{S}_\tau + \mathbf{N}_\tau \tag{7.7}$$

where $\mathbf{Y}_\tau$ is the $n_R \times L_\tau$ received signals matrix, σ_τ^2 is the SNR during the training phase, $\mathbf{S}_\tau$ is the $n_T \times L_\tau$ matrix of training signals sent over L_τ and $\mathrm{Tr}\{\mathbf{S}_\tau \mathbf{S}_\tau^H\} = n_T\, L_\tau$. In data transmission phase,

$$\mathbf{Y}_d = \sqrt{\frac{\sigma_d^2}{n_T}}\,\mathbf{H}\,\mathbf{S}_d + \mathbf{N}_d \tag{7.8}$$

$\mathbf{S}_d$ is the $n_T \times L_d$ matrix of data signals and $\mathrm{E}\big[\mathrm{Tr}\{\mathbf{S}_d \mathbf{S}_d^H\}\big] = n_T\, L_d$ [201]. It is shown in [202] that to estimate the continuous flat-fading MIMO channels, the ML estimator for block-fading model can be utilized to estimate the channel matrix from the received signal $\mathbf{Y}_\tau$ and known training signals $\mathbf{S}_\tau$ [201]. The ML channel estimate for block-fading channels can be obtained by post-multiplying Equation (7.7) by $\mathbf{S}_\tau^H$ as [170]

$$\hat{\mathbf{H}} = \sqrt{\frac{n_T}{\sigma_\tau^2}}\,\mathbf{Y}_\tau \mathbf{S}_\tau^H (\mathbf{S}_\tau \mathbf{C}_\tau^H)^{-1}. \tag{7.9}$$

To obtain a meaningful estimate of $\mathbf{H}$, we need at least as many measurements as unknowns, which implies that $n_R\, L_\tau \geq n_R\, n_T$ or $L_\tau \geq n_T$ [201]. Thus $\hat{\mathbf{H}}$ is independent of the number of n_R.

The optimal training sequence which minimize the *mean square estimation error* (MSE) of Equation (7.9) are mutually orthogonal with respect to time among the transmitter antennas, i.e., $\mathbf{S}_\tau = \sqrt{L_\tau}\Sigma$ where Σ is a matrix with orthonormal columns and $\mathbf{S}_\tau \mathbf{S}_\tau^H = L_\tau\, \mathbf{I}_{n_T}$ where $\mathbf{I}_{n_T}$ is the $n_T \times n_T$ identity matrix [170] [202]. A good choice of orthogonal training sequences is the FFT matrix, i.e. $\mathbf{S}_{m,i} = e^{-2\pi j (m-1)(i-1)/L_\tau}$ where $\mathbf{S}_{m,i}$ is the (m,i)-th element of the training matrix $\mathbf{S}_\tau$, $1 \leq m \leq n_T$, $1 \leq i \leq L_\tau$ [202]. Thus Equation (7.9) can be written as

$$\hat{\mathbf{H}} = \frac{1}{L_\tau}\sqrt{\frac{n_T}{\sigma_\tau^2}}\,\mathbf{Y}_\tau \mathbf{C}_\tau^H \tag{7.10}$$

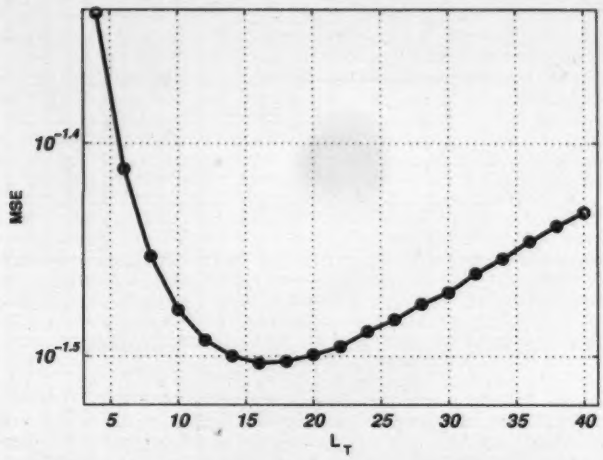

Figure 7.5: Effect of Training Length on the MSE of the Channel.

Therefore, if the the training symbols are orthogonal, no matrix inversion is required at the receiver. While the channel estimate $\hat{\mathbf{H}}$ in (7.10) can be efficiently implemented on FPGAs, effect of various parameters such as training length on the MSE of the channel estimation must be considered. Our initial simulation results show that the channel can be improperly learned if too little training are transmitted and if too much training information is sent then there is no time left for data transmission before the channel changes. It is shown that n_T is the smallest training interval length that guarantees meaningful estimates of the channel matrix [170]. Figure 7.5 plots effect of training length on the MSE of the channel. The MSE due to noise decreases with L_τ but MSE due to temporal variations increases with L_τ and L [202]. In this simulation we used 16-QAM modulation, symbol rate is assumed 1 μs, maximum Doppler frequency is $f_D = 3$ Hz, SNR is 30 dB, and both L_τ and L increase at a fixed ratio, $L_\tau/L = 20\%$.

7.1.4.1 DSP-RAM With Complex Arithmetic

As discussed in Chapter 6, using a real arithmetic processor to solve a complex-valued system leads to an inefficient implementation. For example, it is shown in [175] that complex matrix inversion can be up to twice faster than inversion of a real-valued matrix for $n \geq 3$. As future work, one could consider realizing DSP-RAM with complex arithmetic support for faster and more efficient realization of baseband signal processing. For example, inversion of a complex matrix can be performed using the following scheme [203] more

efficiently than realizing it in real domain. Let $\mathbf{H} = \mathbf{A} + j\mathbf{B}$ be a complex matrix with an inverse $\mathbf{H}^{-1} = \mathbf{A}_1 + j\mathbf{B}_1$, where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{A}_1$, and $\mathbf{B}_1$ are all real. It is shown that $\mathbf{H}^{-1}$ can be obtained by inverting real matrices. If $\mathbf{A}$ is non-singular, then $\mathbf{A}_1$ and $\mathbf{B}_1$ can be written as

$$\begin{aligned} \mathbf{A}_1 &= (\mathbf{A} + \mathbf{B}\mathbf{A}^{-1}\mathbf{B})^{-1} \\ \mathbf{B}_1 &= -\mathbf{A}_1\mathbf{B}\mathbf{A}^{-1}. \end{aligned}$$

Similarly, if $\mathbf{B}$ has an inverse, then the solution can be written as

$$\begin{aligned} \mathbf{B}_1 &= -(\mathbf{B} + \mathbf{A}\mathbf{B}^{-1}\mathbf{A})^{-1} \\ \mathbf{A}_1 &= -\mathbf{B}_1\mathbf{A}\mathbf{B}^{-1} \end{aligned}$$

This technique can be used for the efficient implementation of a matrix inversion, the key operation in decoding of received symbols in MIMO systems.

Bibliography

[1] A. J. Paulraj, D. Gesbert, and C. Papadias. *Smart Antennas for Mobile Communications*. Encyclopedia For Electrical Engineering, John Wiley Publishing Co., 2000.

[2] A. Paulraj, R. Nabar, and D. Gore. *Introduction to Space-Time Wireless Communications*. Cambridge University Press, 2003.

[3] G. J. Foschini. Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas. *Bell System Technical Journal*, 1(2):41–59, Autumn 1996.

[4] A. Alimohammad and B. F. Cockburn. An efficient parallel architecture for implementing LST decoding in MIMO systems. *IEEE Transactions on Signal Processing*, 54(10):3899–3907, 2006.

[5] M. R. Sturgill, G. Cortez, R. Avinun, and S. M. Alamouti. Design and verification of third generation wireless communication systems. Technical Report Article, CoWare, Inc., December 2003.

[6] Agilent Technologies Inc. *Baseband Studio for Fading*, 2005.

[7] Rohde & Schwarz. *Baseband Fading Simulator ABFS, Reduced costs through baseband simulation*, 1999.

[8] C. S. Patel, G. L. Stüber, and T. G. Pratt. Comparative analysis of statistical models for the simulation of Rayleigh faded cellular channels. *IEEE Transactions on Communications*, 53:1017–1026, 2005.

[9] A. Alimohammad, B. F. Cockburn, and C. Schlegel. A Compact and Accurate Gaussian Variate Generator. *submitted to IEEE Transactions on VLSI Systems*.

[10] A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete Time Signal Processing*. Prentice Hall, 1999.

[11] S. Haykin and M. Moher. *Introduction to Analog and Digital Communications*. John Wiley and Sons Canada, Ltd., 2006.

[12] B. Sklar. *Digital Communications: Fundamentals and Applications*. Prentice Hall, 2001.

[13] M. C. Jeruchim, P. Balaban, and K. S. Shanmugan. *Simulation of communication systems : modeling, methodology, and techniques*. New York: Kluwer Academic Publishers, 2000.

[14] A. Papoulis and S. U. Pillai. *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, 2002.

[15] J. G. Proakis. *Digital Communications*. McGraw-Hill, 2001.

[16] W. C. Jakes. *Microwave Mobile Communications*. Piscataway, NJ: Wiley-IEEE Press, 1974.

[17] G. S. Prabhu and P. M. Shankar. Simulation of flat fading using MATLAB for classroom instruction. *IEEE Transactions on Education*, 45(1):19–25, 2002.

[18] P. M. Shankar. *Introduction to Wireless Systems*. John Wiley & Sons, Inc., 2001.

[19] W. H. Tranter, K. Sam Shanmugan, T. S. Rappaport, and K. L. Kosbar. *Principles of Communication Systems Simulation with Wireless Applications*. Prentice Hall, 2003.

[20] H. Hashemi. The indoor radio propagation channel. *Proceedings of the IEEE*, 81(7):943–968, July 1993.

[21] D. C. Cox. 910 MHz urban mobile radio propagation: multipath characteristics in New York city. *IEEE Transactions on Vehicular Technology*, 22(4):104–109, 1973.

[22] G. L. Turin *et al.* A statistical model of urban multipath propagation. *IEEE Transactions on Vehicular Technology*, 21(1):1–11, 1972.

[23] G. L. Stüber. *Principles of Mobile Communication*. New York: Kluwer Academic Publishers, 2001.

[24] R. H. Clarke. A statistical theory of mobile-radio reception. *Bell System Technical Journal*, 47:957–1000, 1968.

[25] M. J. Gans. A power spectral theory of propagation in the mobile radio environment. *IEEE Transactions on Vehicular Technology*, 21(1):27–38, 1972.

[26] W. C. Y. Lee. *Mobile Communications Engineering*. New York: McGraw Hill, 1982.

[27] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

[28] P. Hoeher. A statistical discrete-time model for the WSSUS multipath channel. *IEEE Transactions on Vehicular Technology*, 41:461–468, 1992.

[29] P. Dent, G. E. Bottomley, and T. Croft. Jakes fading model revisited. *Electronics Letters*, 29(13):1162–1163, June 1993.

[30] W. C. Jakes. *Microwave Mobile Communications*. Piscataway, NJ: Wiley-IEEE Press, 1994.

[31] M. Pätzold, U. Killat, F. Laue, and Y. Li. On the statistical properties of deterministic simulation models for mobile fading channels. *IEEE Transactions on Vehicular Technology*, 47:254–269, 1998.

[32] M. Pätzold and F. Laue. Statistical properties of Jakes' fading channel simulator. In *Proceedings of IEEE Vehicular Technology Conference*, pages 712–718, 1998.

[33] M. Pätzold, R. Garcia, and F. Laue. Design of high-speed simulation models for mobile fading channels by using table look-up techniques. *IEEE Transactions on Vehicular Technology*, 49(4):1178–1190, July 2000.

[34] M. F. Pop and N. C. Beaulieu. Limitations of sum-of-sinusoids fading channel simulators. *IEEE Transactions on Communications*, 49:699–708, 2001.

[35] Y. Li and X. Huang. The simulation of independent Rayleigh faders. *IEEE Transactions on Communications*, 50(9):1503–1514, September 2002.

[36] Y. R. Zheng and C. Xiao. Improved models for the generation of multiple uncorrelated Rayleigh fading waveforms. *IEEE Communications Letters*, 6:256–258, 2002.

[37] C. Xiao, Y. R. Zheng, and N. Beaulieu. Statistical simulation models for Rayleigh and Rician fading. In *Proceedings of IEEE International Communication Conference*, pages 3524–3529, 2003.

[38] Y. R. Zheng and C. Xiao. Simulation models with correct statistical properties for Rayleigh fading channels. *IEEE Transactions on Communications*, 51:920–928, 2003.

[39] C. A. G. de Leon, M. C. Bean, and J. S. Garcia. Generation of correlated Rayleigh-fading envelopes for simulating the variant behavior of indoor radio propagation channels. In *Proceedings of IEEE Vehicular Technology Conference*, pages 4245–4249, 2004.

[40] G. Rekaya and J. C. Belfiore. On the complexity of ML lattice decoders for decoding linear full rate space-time codes. In *IEEE International Symposium on Information*

Theory, page 206, 2003.

[41] J. Hennessy and D. A. Patterson. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann Publishers, 2003.

[42] B. Hassibi and H. Vikalo. On the expected complexity of sphere decoding. In *Proceedings of Conference on Signals, Systems and Computers*, pages 1051–1055, 2001.

[43] J. P. Shen and M. Lipasti. *Modern Proccessor Design: Fundamentals of Superscalar Processors*. McGraw-Hill Companies, 2003.

[44] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation : Numerical Methods*. Prentice-Hall, 1997.

[45] Texas Instruments. *TMS320C64x Technical Overview*, 2001.

[46] Texas Instruments. *TMS320C5x DSP Design Workshop Student Guide*, 1997.

[47] S. Rajagopal and S. Rixner J. R. Cavallaro. A programmable baseband processor design for software defined radios. In *Midwest Symposium on Circuits and Systems*, pages III–413–III–416, 2002.

[48] Altera. *The Stratix II GX, Device Handbook,*, 2006.

[49] Xilinx. *Virtex-II Pro™ Platform FPGAs: Functional Description*, January 2003.

[50] Analog Devices Inc. *TigerSHARC Embedded Processor*, 2003.

[51] J. M. Rabaey. Silicon platforms for the next generation wireless systems - What role does reconfigurable hardware play? In *International Conference on Field-Programmable Logic and Applications*, pages 277–285, 2002.

[52] R. Baines. The DSP bottleneck. *IEEE Communications Magazine*, 33(5):46–54, May 1995.

[53] H. Keding, M. Coors, O. Luthje, and H. Meyr. Fast bit-true simulation. In *IEEE Design Automation Conference*, pages 708 – 713, 2001.

[54] M. Rupp, A. Burg, and E. Beck. Rapid prototyping for wireless designs: the five-ones approach. *International European Association for Signal Processing Journal*, 83(7):1427–1444, July 2003.

[55] C. S. Petrie and J. A. Connelly. The sampling of noise for random number generation. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 26–29, 1999.

[56] S. Rocchi and V. Vignoli. A chaotic CMOS true-random analog/digital withe noise generator. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 463–466, 1999.

[57] H. Zhun and H. Chen. A truly random number generator based on thermal noise. In *Proceedings of IEEE International Conference on ASIC*, pages 862–864, 2001.

[58] G. S. Muller and C. K. Pauw. On the generation of a smooth Gaussian random variable to 5 standard deviations. In *Proceedings of IEEE Southern African Conference on Communications and Signal Processing*, pages 62–66, 1988.

[59] F. C. Ionescu. Theory and practice of a fully controllable white noise generator. In *Proceedings of IEEE International Semiconductor Conference*, pages 319–322, 1996.

[60] K. Tae, J. Chung, and D. Kim. Noise generation system using DCT. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 29–32, 2002.

[61] E. Boutillon, J. L. Danger, and A. Gazel. Design of high speed AWGN communication channel emulator. In *Analog Integrated Circuits and Signal Processing*, pages 133–142, 2003.

[62] D.-U. Lee, W. Luk, J. D. Villasenor, and P. Y. K. Cheung. A Gaussian noise generator for hardware-based simulations. *IEEE Transactions on Computers*, 53(12):1523–1534, December 2004.

[63] P. Kohlbrenner, L. Martin, and K. Gaj. An embedded true random number generator for FPGAs. In *International Symposium on Field Programmable Gate Arrays*, pages 71–78, 2004.

[64] G. E. P. Box and M. E. Muller. A note on the generation of random normal deviates. *The Annals of Math. Statistics*, 29:610–611, 1958.

[65] P. Hellekalek. Good random number generators are (not so) easy to find. In *Second IMACS Symposium on Mathematical Modelling*, pages 485–505, 1998.

[66] W. Hörmann and J. Leydold. Continuous random variate generation by fast numerical inversion. *ACM Transactions on Modeling and Computer Simulation*, 13(4):347–362, 2003.

[67] J. Chen, J. Moon, and K. Bazargan. Reconfigurable readback-signal generator based on a field-programmable gate array. *IEEE Transactions on Magnetics*, 40(3):1744–1750, 2004.

[68] M. E. Muller. A comparison of methods for generating normal deviates on digital computers. *Journal of the ACM*, 6(3):376–383, 1959.

[69] G. Marsaglia and W. W. Tsang. The Ziggurat method for generating random variables. *Journal of Statistical Software*, 5(8):1–7, 2000.

[70] G. Marsaglia and W. W. Tallahassee. A simple method for generating gamma variables. *ACM Transactions on Mathematical Software*, 26(3):363–372, 2000.

[71] Y. Fan, Z. Zilic, and M. W. Chiang. A versatile high speed bit error rate testing scheme. In *Proceedings of IEEE International Symposium on Quality Electronic Design*, pages 395–400, 2004.

[72] M. D. Ercegovac and T. Lang. *Digital Arithmetic*. Morgan Kaufmann, 2004.

[73] L. Dong-U *et al.* A hardware Gaussian noise generator for channel code evaluation. In *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 69–78, 2003.

[74] A. A. Kalatchikov and G. G. Streltsov. FPGA implementation of Gaussian noise generator. In *5th Annual International Siberian Workshop on Electron Devices and Materials*, 2004.

[75] Xilinx Inc., San Jose, CA. *Additive White Gaussian Noise (AWGN) Core v1.0*, 2002.

[76] A. Alimohammad, B. F. Cockburn, and C. Schlegel. An iterative hardware Gaussian noise generator. In *Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 649–652, August 2005.

[77] A. Alimohammad, B. F. Cockburn, and C. Schlegel. Area-efficient parallel white Gaussian noise generator. In *Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering*, pages 1855–1858, 2005.

[78] R.P. Brent. Fast normal random number generators for vector processors. Technical Report Technical Report TR-CS-93-04, The Australian National University, July 1993.

[79] D. Lee *et al.* A hardware Gaussian noise generator using the Wallace method. *IEEE Transactions on VLSI Systems*, 13(8):911–920, 2005.

[80] I. Vattulainen, T. Ala-Nissila, and K. Kankaala. Physical models as tests of randomness. *Physical Review E (Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics)*, 52(3):3205–3214, 1995.

[81] P. D. Coddington. Tests of random number generators using Ising model simulations.

In *Proceedings of the US-Japan Bilateral Seminar on New Trends in Computer Simulations of Spin Systems*, 1996.

[82] A. Compagner. Operational conditions for random-number generation. *Phys. Review E 52*, 52(5):5634–5645, 1995.

[83] J. Gentle *et al.* *Handbook of Computational Statistics*. Springer-Verlag, 2004.

[84] P. L'Ecuyer. *Random Number Generation, Chapter 3 of Elsevier Handbooks in Operations Research and Management Science: Simulation*. Elsevier Science, 2006.

[85] J. Grodendorst *et al.* Pseudo random number generation and quality checks. *Quantum Simulations of Complex Many-Body Systems, Lecture Notes*, 10:447–458, 2002.

[86] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C Example Book : The Art of Scientific Computing*. Cambridge University Press, 1992.

[87] P. D. Hortensius, R. D. McLeod, and H. C. Card. Parallel random number generation for VLSI systems using cellular automata. *IEEE Transactions on Computers*, 38(10):1466–1473, 1989.

[88] J. Lindholm. An analysis of the pseudo-randomness properties of subsequences of long m-sequences. *IEEE Transactions on Information Theory*, 14(4):569–576, 1968.

[89] Xilinx. *Using the Virtex Look-Up Tables, The Quarterly Journal for Xilinx Programmable Logic Users*, Xcell Journal, 2000.

[90] P. P. Chu and R. E. Jones. Design techniques of FPGA based random number generator. In *Military and Aerospace Applications of Programmable Devices and Technologies Conference*, 1999.

[91] S. Wolfram. Random sequence generation by cellular automata. *Advances in Applied Mathematics*, 7(10):123–169, 1986.

[92] P. H. Bardell. Analysis of cellular automata used as pseudorandom pattern generators. In *Proceedings of International Test Conference*, pages 762–768, 1990.

[93] G. Marsaglia, A. Zaman, and W. W. Tsang. Towards a universal random number generator. *Stat. Prob. Letters*, 8:35–39, 1990.

[94] P. L'Ecuyer and F. Panneton. A new class of linear feedback shift register generators. In *Proceedings of the 2000 Winter Simulation Conference*, pages 690–696, 2000.

[95] R. C. Tausworthe. Random numbers generated by linear recurrence modulo two.

Mathematics and Computation, 19:201–209, 1965.

[96] P. L'Ecuyer and J. Granger-Pich. Combined Generators with Components from Different Families. *Mathematics and Computers in Simulation*, 62:395–404, 2003.

[97] P. L'Ecuyer. Tables of maximally equidistributed combined LFSR generators. *Mathematics of Computation archive*, 68(225):261–269, 1999.

[98] P. L'Ecuyer. Maximally Equidistributed Combined Tausworthe Generators. *Mathematics of Computation*, 65(213):203–213, 1996.

[99] P. Hellekalek. Inversive pseudorandom number generators: concepts, results, and links. In *Proceedings of the Winter Simulation Conference*, pages 255–262, 1995.

[100] P. Hellekalek and S. Wegenkittl. Empirical evidence concerning AES. *ACM Transactions on Modeling and Computer Simulation*, 13(4):322–333, 2003.

[101] A. Hodjat and I. Verbauwhede. A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA. In *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 308–309, 2004.

[102] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. U.S. Government Printing Office, 1972.

[103] Synopsys. *DesignWare IP Family Reference Guide*, 2005.

[104] F. de Dinechin and A. Tisserand. Multipartite Table Methods. *IEEE Transactions on Computers*, 54(3):319–330, 2005.

[105] B. Parhami. *Computer Arithmetic, Algorithms and Hardware Design*. Oxford University Press, 2000.

[106] J.E. Volder. The CORDIC Trigonometric Computing Technique. *IRE Transactions on Electronic Computers*, EC-8(3):3.4.1, 1959.

[107] Xilinx. *Xilinx Cordic LogiCore Product Specification*, April, 2005.

[108] N. Chernov, C. Lesort, and N. Simanyi. On the complexity of curve fitting algorithms. *Journal of Complexity*, 20(4):484–492, August 2004.

[109] M. Schmookler and K. Nowka. Leading zero anticipation and detection - A comparison of methods. In *Proceedings of IEEE Symposium on Computer Arithmetic*, pages 7–12, 2001.

[110] V. G. Oklobdzija. An implementation algorithm and design of a novel leading zero detector circuit. In *in Proceedings Asilomar Conference on Signals, Systems and*

Computers", pages 391–395, 1992.

[111] R. B. D'Agostino and M. A. Stephens. *Goodness-of-Fit Techniques*. Marcel Dekker Inc., 1986.

[112] H. R. Neave. On using the Box-Muller transformation with multiplicative congruential pseudorandom number generators. *Appl. Stat.*, 22:92–97, 1973.

[113] P. Bello. Characterization of randomly time-variant linear channels. *IEEE Transactions on Communications*, 11(4):360–393, 1963.

[114] J. I. Smith. A computer generated multipath fading simulation for mobile radio. *IEEE Transactions on Vehicular Technology*, 24(3):39–40, August 1975.

[115] T. S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, 2002.

[116] S. O. Rice. Mathematical analysis of random noise. *Bell System Technical Journal*, 23:282–332, 1944.

[117] E. F. Casas and C. Leung. A simple digital fading simulator for mobile radio. In *IEEE Vehicular Technology Conference*, pages 212–217, 1988.

[118] P. J. Cullen, P. C. Fannin, and S. S. Swords. DSP implementation of a wideband frequency selective fading simulator. In *International Conference on Antennas and Propagation*, pages 492 – 495, 1991.

[119] J. F. An, A. M. D. Turkmani, and J. D. Parsons. Implementation of a DSP-based frequency non-selective fading simulator. In *International Conference on Radio Receivers and Associated Systems*, pages 20–24, 1990.

[120] M. Lecours and F. Marceau. Design and implementation of a channel simulator for wideband mobile radio transmission. In *IEEE Vehicular Technology Conference*, pages 652–655, 1989.

[121] A. K. Salkintzis. Implementation of a digital wide-band mobile channel simulator. *IEEE Transactions on Broadcasting*, 45(1):122 – 128, 1999.

[122] W. Zhuang. Adaptive importance sampling for bit error rate estimation inslowly fading channels. In *IEEE International Symposium on Wireless Networks*, pages 1330–1334, 1994.

[123] M. A. Wickert and J. Papenfuss. Implementation of a real-time frequency-selective RF channel simulator using a hybrid DSP-FPGA architecture. *IEEE Transactions on Microwave Theory and Techniques*, 49(8):1390 – 1397, 2001.

[124] D. Derrien and E. Boutillon. Quality Measurement of a Colored Gaussian Noise Generator Hardware Implementation Based on Statistical Properties. In *Proceedings of the IEEE International Symposium on Signal Processing and Information Technology*, 2002.

[125] G. Arredondo, W. Chriss, and E. Walker. A multipath fading simulator for mobile radio. *IEEE Transactions on Communications*, 21(11):1325–1328, November 1973.

[126] R. N. Kolte, S. C. Kwatra, and G. H. Stevens. Computer controlled hardware simulation of fading channel models. In *Proceedings of IEEE International Conference on Communications*, pages 1646–1650, 1988.

[127] A. Verschoor, A. Kegel, and J. C. Arnbak. Hardware fading simulator for a number of narrowband channels with controllable mutual correlation. *Electronics Letters*, 24(22):1367–1369, 1988.

[128] E. Casas and C. Leung. A simple digital fading simulator for mobile radio. *IEEE Transactions on Vehicular Technology*, 39(3):205–212, August 1990.

[129] Z. Yansheng. A Rayleigh fading simulator for mobile radio. In *Proceedings of IEEE Conference on Computer, Communication, Control and Power Engineering*, pages 143–145, 1993.

[130] P. Cullen, P. Fannin, and A. Garvey. Real-time simulation of randomly time-variant linear systems: the mobile radio channel. *IEEE Transactions on Instrumentation and Measurement*, 43(4):583–591, 1994.

[131] M. Cui, H. Murata, and K. Araki. FPGA implementation of 4x4 MIMO test-bed for spatial multiplexing systems. In *IEEE International Symposium on Personal Indoor and Mobile Radio Communications*, pages 3045–3048, 2004.

[132] E. Biglieri, J. Proakis, and S. Shamai. Fading channels: information-theoretic and communications aspects. *IEEE Transactions on Information Theory*, 44(6):2619–2692, 1998.

[133] R. N. Kolte, S. C. Kwatra, and G. H. Stevens. Computer controlled hardware simulation of fading channel models. In *Proceedings of IEEE International Conference on Communications*, pages 1646–1650, 1998.

[134] D. J. Young and N. C. Beaulieu. The generation of correlated Rayleigh random variates by inverse Fourier transform. *IEEE Transactions on Communications*, 48:1114–1127, 2000.

[135] E. N. Gilbert. Energy reception for mobile radio. *Bell System Technical Journal*,

pages 1779 – 1803, Oct. 1965.

[136] C. Xiao, Y. R. Zheng, and N. C. Beaulieu. Second-order statistical properties of the WSS Jakes' fading channel simulator. *IEEE Transactions on Communications*, 50(6):888–891, June 2002.

[137] C-X. Wang and M. Pätzold. Methods of generating multiple uncorrelated Rayleigh fading processes. In *Proceedings of IEEE Semiannual Vehicular Technology Conference*, pages 510–514, 2003.

[138] Ascom. *Full Featured Channel Simulator*, 2002.

[139] C. Komninakis. A fast and accurate Rayleigh fading simulator. In *Proceedings of IEEE Global Telecommunications Conference*, pages 3306–3310, 2003.

[140] P. Hoeher and A. Steingaß. Modeling and emulation of multipath fading channels using controlled randomness. In *in Proceedings ITG-Fachtagung "Wellenausbreitung bei Funksystemen und Mikrowellensystemen"*, pages 209–220, 1998.

[141] A. Zajić and G. L. Stüber. Efficient simulation of Rayleigh fading with enhanced decorrelation properties. *to appear in IEEE Transactions on Wireless Communications*.

[142] K. E. Baddour and N. C. Beaulieu. Autoregressive models for fading channel simulation. In *Proceedings of IEEE Global Telecommunications Conference*, pages 1187–1192, 2001.

[143] Spirent Communications. *Wireless Channel Emulator*, 2006.

[144] Japan Radio Co. *Multipath Fading Simulator*, 2005.

[145] NoiseCom, Paramus, NJ. *NoiseCom MP-2500 Multipath Fading Emulator, Technical Manual*, 1996.

[146] J. Carletta, R. Veillette, F. Krach, and Z. Fang. Determining appropriate precisions for signals in fixed-point IIR filters. In *Proceedings of IEEE Design Automation Conference*, pages 656–661, 2003.

[147] L. B. Jackson. *Digital Filters and Signal Processing*. Kluwer Academic Publishers, 1989.

[148] The Mathworks. *Filter Design Toolbox For Use with Matlab, User's Guide*, 2005.

[149] R. A. Goubran, H. M. Hafez, and A. U. H. Sheikh. Implementation of a real-time mobile channel simulator using a DSP chip. *IEEE Transactions on Instrumentation and Measurement*, 40(4):709–714, 1991.

[150] M. S. Lim and H. K. Park. The implementation of the mobile channel simulator in the baseband and its application to the quadrature type GMSK modem design. In *IEEE Vehicular Technology Conference*, pages 469–500, 1990.

[151] G. J. R. Povey, P. M. Grant, and R. D. Pringle. A decision-directed spread-spectrum RAKE receiver for fast-fading mobile channels. *IEEE Transactions on Vehicular Technology*, 45(3):491–502, November 1996.

[152] A. Stéphenne and B. Champagne. Effective multi-path vector channel simulator for antenna array systems. *IEEE Transactions on Vehicular Technology*, 49(6):2370 – 2381, 2000.

[153] A. Antoniou. *Digital Filters, Analysis, Design, and Applications*. McGraw-Hill Book Co., 1993.

[154] G. J. Foschini and M. J. Gans. On limits of wireless communications in a fading environment when using multiple antennas. In *Proceedings of Wireless Personal Communications*, pages 311–335, March 1998.

[155] G. Golden *et al.* Detection algorithm and initial laboratory results using the V-BLAST space-time communication architecture. *Electronics Letters*, 35(1):14–15, January 1999.

[156] S. Loyka and F. Gagnon. Performance analysis of the V-BLAST algorithm: an analytical approach. *IEEE Transactions on Wireless Communications*, 3(4):1326–1337, July 2004.

[157] J. Benesty, Y. Huang, and J. Chen. A fast recursive algorithm for optimum sequential signal detection in a BLAST system. *IEEE Transactions on Signal Processing*, 51(7):1722–1730, July 2003.

[158] B. Hassibi. An efficient square-root algorithm for BLAST. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 737–740, 1999.

[159] D. Wubben *et al.* Efficient algorithm for detecting layered space-time codes. In *4th International ITG Conference on Source and Channel Coding*, pages 399–405, January 2002.

[160] Z. Guo and P. Nilsson. A low-complexity VLSI architecture for square root MIMO detection. In *IASTED Circuits, Signals and Systems*, May 2003.

[161] D. Garrett *et al.* A 28.8 Mb/s 4x4 MIMO 3G high-speed downlink packet access receiver with normalized least mean square equalization. In *IEEE International Solid-*

State Circuits Conference, pages 420–536, February 2004.

[162] A. Adjoudani *et al.* Prototype experience for MIMO BLAST over third-generation wireless system. *IEEE Journal on Selected Areas in Communications*, 21(3):440–451, April 2003.

[163] P. Murphy *et al.* An FPGA based rapid prototyping platform for MIMO systems. In *Thrity-Seventh Asilomar Conference on Signals, Systems and Computers*, pages 900–904, November 2003.

[164] A. Burg *et al.* FPGA implementation of a MIMO receiver front-end for the UMTS downlink. In *International Zurich Seminar on Broadband Communications*, pages 8–1–8–6, February 2002.

[165] S. J. Dillen and B. F. Cockburn. Parallel filtering and thresholding of images on the SIMD DSP-RAM architecture. In *Proceedings of IEEE Canadian Conference of Electrical and Computer Engineering*, pages 995–1000, May 2002.

[166] B. S. H. Kwan, B. F. Cockburn, and D. G. Elliott. Implementation of DSP-RAM: An architecture for parallel digital signal processing. In *IEEE Canadian Conference on Electrical and Computer Engineering*, pages 341–346, May 2001.

[167] M. Gokhale, B. Holmes, and K. Iobst. Processing in memory: The Terasys massively parallel PIM array. *Computer*, 28(4):23–31, April 1995.

[168] A. Chandrakasan, S. Sheng, and R. Brodersen. Low-power CMOS Digital Design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, April 1992.

[169] T. L. Marzetta and B. M. Hochwald. Capacity of a mobile multiple-antenna communication link in a rayleigh flat-fading environment. *IEEE Transactions on Infomation Theory*, 45(1):139–157, 1999.

[170] T. L. Marzetta. BLAST Training: Estimating channel characteristics for high capacity space-time wireless. In *Proceedings of the 37th Annual Allerton Conference on Communication, Control, and Computing*, pages 958–966, 1999.

[171] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[172] G. Strang. *Linear Algebra and its Applications*. Saunders, 1988.

[173] J. Demmel, B. Diament, and G. Malajovich. On the complexity of computing error bounds. *Foundations of Computational Mathematics*, 1(1):101–125, 2001.

[174] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University

Press, 1996.

[175] L. W. Ehrlich. Complex matrix inversion versus real. *Communications of the ACM*, 13(9):561–562, 1970.

[176] P. W. Wolniansky, G. J. Foschini, G. D. Golden, and R. A. Valenzuela. V-BLAST: An architecture for realizing very high data rates over the rich scattering wireless channel. In *Proceedings of Int. Symposium on Signals, Systems, and Electronics*, pages 295–300, 1998.

[177] M. O. Damen, K. Abed-Meraim, and S. Burykh. Iterative QR detection for BLAST. *Wireless Personal Communications*, 19(3):179–192, December 2001.

[178] M. O. Damen, H. El Gamal, and G. Caire. On maximum likelihood detection and the search for the closest lattice point. *IEEE Transactions Information Theory, Special issue on space-time coding*, 2003.

[179] D. Gesbert and J. Akhtar. Breaking the barriers of Shannon's capacity: An overview of MIMO wireless systems. *Telektronikk Telenor Journal*, January 2002.

[180] E. Viterbo and E. Biglieri. A universal lattice decoder. In *in 14 Collog. GRETSI colloque*, pages 611–614, 1993.

[181] E. Viterbo and J. Boutros. A universal lattice code decoder for fading channels. *IEEE Transactions on Information Theory*, 45(5):1639–1642, July 1999.

[182] M. O. Damen, A. Chkeif, and J.-C. Belfiore. Lattice code decoder for space-time codes. *IEEE Communications Letters*, 4(5):161–163, May 2000.

[183] M. O. Damen, K. Abed-Meraim, and J.-C. Belfiore. Generalized sphere decoder for asymmetrical space-time communication architecture. *Electronics Letters*, 36(2):166–167, January 2000.

[184] U. Fincke and M. Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation*, 44(170):463–471, 1985.

[185] D. Samardzija and N. Mandayam. Pilot-assisted estimation of MIMO fading channel response and achievable data rates. *IEEE Transactions on Signal Processing*, 51(11):2882–2890, November 2003.

[186] S. Bäro, G. Bauch, A. Pavlic, and A. Semmler. Improving BLAST performance using space-time block codes and turbo decoding. In *Proceedings of Globecom*, 2000.

[187] M. C. Pease. Matrix inversion using parallel processing. *Journal of the ACM*, 4(14):757–764, 1967.

[188] K. Diefendoriff. PC Processor Microarchitecture. *Microdesign Resources, Microprocessor Report*, pages 1–7, July 1999.

[189] N. Yamashita *et al.* A 3.84 GIPS Integrated Memory Array Processor with 64 Processing Elements and a 2-Mb SRAM. *IEEE Journal of Solid-State Circuits*, 29(11):1336–1343, November 1994.

[190] Christoforos Kozyrakis. *Scalable Vector Media-processors for Embedded Systems*. PhD thesis, University of California - Berkeley, 2002.

[191] Z. Wang, B. F. Cockburn, D. G. Elliott, and W. A. Krzymien. DSP-RAM: A Logic-Enhanced Memory Architecture for Communication Signal Processing. *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 475–478, August 1999.

[192] Z. Guo and P. Nilsson. A VLSI implementation of MIMO detection for future wireless communications. In *IEEE Proceedings on Indoor and Mobile Radio Communications*, pages 2852–2856, 2003.

[193] Intel. *Intel AP-928: Streaming SIMD Extensions*, 1999.

[194] ARM. *ARM7TDMI Technical Reference Manual*, 2001.

[195] Intel. *Intel PXA255 Processor: Developer's Manual*, 2004.

[196] Texas Instruments. *TMS320C6000 CPU and Instruction Set Reference Guide*, 2000.

[197] TSMC. *TSMC 0.18-μm process technology*, 2003.

[198] H.-Y Lo and J.-L. Chen. A hardwired generalized algorithm for generating the logarithm base-k by iteration . *IEEE Transactions on Computers*, 36(11):1363–1367, 1987.

[199] M. Pätzold, U. Killat, F. Laue, and Y. Li. A new and optimal method for the derivation of deterministic simulation models for mobile radio channels. In *IEEE Vehicular Technology Conference*, pages 1423–1427, 1996.

[200] M. Kiessling and J. Speidel. Statistical transmit processing for enhanced MIMO channel estimation in presence of correlation. In *IEEE Global Telecommunications Conference*, pages 2411 – 2415, 2003.

[201] B. Hassibi and B. Hochwald. Optimal training in space-time systems. In *Conference*

Record of the Thirty-Fourth Asilomar Conference on Signals, Systems and Computers, pages 743–47, 2000.

[202] Q. Sun, D. C. Cox, H. C. Howard, and A. Lozano. Estimation of continuous flat fading MIMO channels. *IEEE Transactions on Wireless Communications*, 1(4):549–553, 2002.

[203] L. Tornheim. Inversion of a complex matrix. *Communications of the ACM*, 4(9):398–398, 1961.

Appendix A

List of Publications Arising From Thesis

United States Provisional Patent Application

[1] A. Alimohammad, S. Fouladi Fard, B. F. Cockburn and C. Schlegel, "Signal filtering and filter design techniques," January, 2007.

Published and Submitted Journal Articles

[1] A. Alimohammad and B. F. Cockburn, "An efficient parallel architecture for implementing LST decoding in MIMO systems," *IEEE Transactions on Signal Processing*, vol. 54, no. 10, Oct. 2006, pp. 3899-3907.

[2] A. Alimohammad, B. F. Cockburn and C. Schlegel, "A compact and accurate Gaussian variate generator," submitted on September 19, 2006 to *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*

[3] A. Alimohammad and B. F. Cockburn, "Modeling and hardware implementation aspects of fading channel simulators," submitted on Aug. 9, 2006 to *IEEE Transactions on Vehicular Technology.*

[4] A. Alimohammad, S. Fouladi Fard, B. F. Cockburn and C. Schlegel, "A compact single-FPGA fading channel simulator," submitted on Nov. 3, 2006 to *IEEE Transactions on Circuits and Systems Part II: Express Briefs.*

Articles in Revision

[1] A. Alimohammad, S. Fouladi Fard, B. F. Cockburn and C. Schlegel, "An accurate SOS-based fading channel emulator," to be submitted to *IEEE Communications Letters.*

[2] S. Fouladi Fard, A. Alimohammad, B. F. Cockburn, C. Schlegel,"Theory and practice of rapid baseband simulation of non-isotropic Rayleigh fading," to be submitted to *IEEE Transactions on Signal Processing.*

[3] A. Alimohammad, Steven J. Dillen and B. F. Cockburn, DSP-RAM: A SIMD processor-in-memory for signal processing, *in revision.*

[4] A. Alimohammad and B. F. Cockburn, A parallelizable displaced sphere decoding method for MIMO systems, *in revision.*

Published and Accepted Conference Papers

[1] A. Alimohammad and B. F. Cockburn, "Compact implementation of a sum-of-sinusoids Rayleigh fading channel simulator," *6th IEEE International Symposium on Signal Processing and Information*

Technology (ISSPIT 2006), Aug. 27-30, 2006.

[2] A. Alimohammad and B. F. Cockburn, "A reconfigurable SOS-based Rayleigh fading channel simulator," *IEEE 2006 International Workshop on Signal Processing Systems* (SIPS 2006), Oct. 2-4, 2006.

[3] A. Alimohammad, B. F. Cockburn, C. Schlegel, "An iterative hardware Gaussian noise generator," *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, August 24-26, 2005, pp. 649-652.

[4] A. Alimohammad, B. F. Cockburn, C. Schlegel, "Area-efficient parallel white Gaussian noise generator," *IEEE Canadian Conference on Electrical and Computer Engineering*, May 1-4, 2005, pp. 1855-1858.

[5] A. Alimohammad, S. Fouladi Fard, B. F. Cockburn, C. Schlegel, "Statistical measurements of random signals on FPGAs," *IEEE Canadian Conference on Electrical and Computer Engineering*, April 22-26, 2007.

[6] S. Fouladi Fard, A. Alimohammad, M. Khorasani, B. F. Cockburn, C. Schlegel,"A compact and accurate FPGA based non-isotropic fading channel simulator," *IEEE Canadian Conference on Electrical and Computer Engineering*, April 22-26, 2007.

Submitted Conference Papers

[1] A. Alimohammad, S. Fouladi Fard, B. F. Cockburn and C. Schlegel, "A flexible filter processor for fading channel emulation," submitted to *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2007.

[2] S. Fouladi Fard, A. Alimohammad, C. Schlegel, B. F. Cockburn, "A high-throughput systolic detector for MIMO systems," submitted to *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2007.

[3] A. Alimohammad, B. F. Cockburn and C. Schlegel, "A compact fading channel simulator using timing-driven resource sharing," submitted to *IEEE International Symposium on Circuits and Systems* (ISCAS), 2007.